Strong simulation: Capturing topology in graph pattern matching

A

Strong Simulation: Capturing Topology in Graph Pattern Matching

Shuai Ma[1] Yang Cao[1] Wenfei Fan[1,2] Jinpeng Huai[1] Tianyu Wo[1]
[1]SKLSDE Lab, Beihang University [2]University of Edinburgh

Graph pattern matching is to find all matches in a data graph for a given pattern graph, and it is often defined in terms of subgraph isomorphism, an NP-complete problem. To lower its complexity, various extensions of graph simulation have been considered instead. These extensions allow graph pattern matching to be conducted in cubic-time. However, they fall short of capturing the topology of data graphs, *i.e.,* graphs may have a structure drastically different from pattern graphs they match, and the matches found are often too large to understand and analyze. To rectify these problems, this paper proposes a notion of *strong simulation*, a revision of graph simulation, for graph pattern matching. (1) We identify a set of criteria for preserving the topology of graphs matched. We show that strong simulation preserves the topology of data graphs and finds a bounded number of matches. (2) We show that strong simulation retains the same complexity as earlier extensions of graph simulation, by providing a cubic-time algorithm for computing strong simulation. (3) We present the locality property of strong simulation, which allows us to develop an effective distributed algorithm to conduct graph pattern matching on distributed graphs. (4) We experimentally verify the effectiveness and efficiency of these algorithms, using both real-life and synthetic data.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous—*Graph pattern matching*; H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: Strong simulation, Dual simulation, Graph simulation, Subgraph isomorphism, Data locality

ACM Reference Format:
Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo, 2014, Strong Simulation: Capturing Topology in Graph Pattern Matching. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 45 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Graph pattern matching is being increasingly used in a number of applications, *e.g.,* software, biology, social networks and intelligence analysis [Liu et al. 2006; Sprinzak et al. 2003; Tian and Patel 2008; Tong et al. 2007; Zou et al. 2009]. Given a pattern graph Q and a data graph G, it is to find all subgraphs of G that match Q. Here matching is typically defined in terms of *subgraph isomorphism* (see, *e.g.,* [Aggarwal and Wang 2010; Gallagher 2006] for surveys): a subgraph G_s of G *matches* Q if there exists a *bijective function* f from the nodes of Q to the nodes in G_s such that (1) for each pattern node u in Q, u and $f(u)$ have the same label, and (2) there exists an edge (u, u') in Q if and only if there exists an edge $(f(u), f(u'))$ in G_s.

Author's addresses: S. Ma, Y. Cao, J. Huai and T. Wo (contact author), School of Computer Science and Engineering, Beihang University, Beijing, China; W. Fan, School of Informatics, Laboratory for Foundations of Computer Science, Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, Scotland, UK.

DOI: http://dx.doi.org/10.1145/0000000.0000000

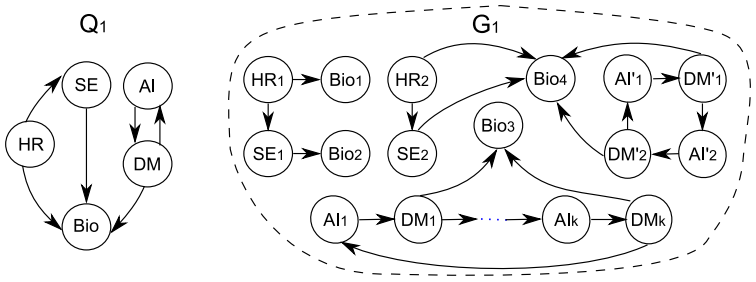


Fig. 1. Social matching: pattern and data graphs

However, subgraph isomorphism is an NP-complete problem [Ullmann 1976]. Moreover, there are possibly exponentially many subgraphs in G that match Q. In addition, as already observed in [Brynielsson et al. 2010; Fan et al. 2010a], subgraph isomorphism is often too restrictive to catch sensible matches, as it requires matches to have exactly the same topology as a pattern graph. However, the structures of real-life graphs are frequently updated with minor adjustments, and the interconnections of the same entities in various datasets may be different [Abiteboul 1997; Fard et al. 2012; Khan et al. 2013]. Worse still, it is common to find information incomplete (*e.g.,* unintentional or intentional hidden links) in social networks [Liben-Nowell and Kleinberg 2003; Chen et al. 2011]. These hinder the applicability of subgraph isomorphism to seek exact matches in emerging applications such as social network analysis, crime detection, protein–protein interaction analysis and software plagiarism detection.

To reduce the complexity, *graph simulation* [Milner 1989] has been adopted for pattern matching. A graph G matches a pattern Q via *graph simulation* if there exists a binary *relation* $S \subseteq V_Q \times V$, where V_Q and V are the set of nodes in Q and G, respectively, such that (1) for each $(u, v) \in S$, u and v have the same label; and (2) for each node u in Q, there exists v in G such that (a) $(u, v) \in S$, and (b) for each edge (u, u') in Q, there exists an edge (v, v') in G such that $(u', v') \in S$. Graph simulation can be computed in quadratic time [Henzinger et al. 1995]. Recently this notion has been extended by mapping edges in Q to (bounded) paths in G [Fan et al. 2010a; Fan et al. 2011], with a cubic-time complexity, to identify matches in, *e.g.,* social networks.

Nevertheless, the low complexity comes at a price: (1) simulation and its extensions [Fan et al. 2010a; Fan et al. 2011] do not preserve the topology of data graphs; in other words, they may match a graph G and a pattern Q with drastically different structures. (2) The match relation S is often too large to understand and analyze. We illustrate these with an example below.

Example 1.1: Consider a real-life example taken from [Terveen and McDonald 2005]. A headhunter wants to find a biologist (Bio) to help a group of software engineers (**SE**s) analyze genetic data. To do this, she uses an expertise recommendation network G_1, as depicted in Fig. 1. In G_1, a node denotes a person labeled with expertise, an edge indicates recommendation, *e.g.,* HR_1 recommends Bio_1, and there is an edge from each DM_i to Bio_3 (not all edges are explicitly shown). The biologist Bio needed is specified with a pattern graph Q_1, also shown in Fig. 1. Intuitively, the Bio has to be recommended by: (a) an HR person since the headhunter trusts the judgment of a person with the same occupation; (b) an SE, *i.e.,* the Bio has experience working with SEs, which makes the Bio easy to communicate with SEs; and (c) a data mining specialist (DM), as data mining techniques are required for the job. To further increase incredibility, (d) the SE is also recommended by an HR person, and (e) there is an artificial intelligence expert (AI) who recommends the DM and is recommended by a DM.

When subgraph isomorphism is used, no match can be found, *i.e.,* no subgraph of G_1 is isomorphic to Q_1. In other words, subgraph isomorphism imposes too strict a constraint on the topology of the graphs matched.

When graph simulation or its extensions [Fan et al. 2010a; Fan et al. 2011] are adopted, *all* four biologists in G_1 are matches for Bio in Q_1. However, Bio_1 and Bio_2 are recommended by either HR only or by SE only in G_1, and Bio_3 is recommended by neither an HR nor an SE. Hence these are not the ones that the headhunter really wants. Only Bio_4 satisfies all these conditions and makes a good candidate.

This tells us that simulation and its extensions [Fan et al. 2010a; Fan et al. 2011] do not preserve the structural properties in graph pattern matching and therefore, may return excessive "matches" that one does not want. Indeed, observe the following.

Topological structure. (a) While Q_1 is a connected graph, G_1 is disconnected, but G_1 matches Q_1 via graph simulation. (b) Node Bio in Q_1 has three "parents", but it matches nodes Bio_1 and Bio_2 in G_1 that have a single "parent" each. (c) The directed cycle with only two nodes AI and DM in Q_1 matches the long cycle consisting of $2k$ nodes, *e.g.,* $\mathsf{AI}_1, \mathsf{DM}_1, \ldots, \mathsf{AI}_k, \mathsf{DM}_k$ and AI_1, in G_1. (d) The undirected cycle with nodes HR, SE and Bio in Q_1 matches the tree rooted at HR_1 in G_1.

Match results. The match relation of graph simulation, when represented as a result graph as suggested in [Fan et al. 2010a], is the entire graph G_1. In general, the result graphs are often large when matching is performed on real-life networks, *e.g.,* LinkedIn[1], which has 19.5M users and yields a graph of 100GB in size. These make it hard to analyze the match results . □

These suggest that we revise the notion of graph simulation to strike a balance between its computational complexity and its ability to capture the topology of graphs. Indeed, graph simulation was proposed as a process algebra to mimic steps of a process [Milner 1989]. To make practical use of it in graph pattern matching, we need to enhance it by incorporating more topological structure of graphs.

Contributions & Roadmap. We introduce a revision of graph simulation that preserves the topology of graphs and has the same complexity as earlier extensions [Fan et al. 2010a; Fan et al. 2011] of graph simulation.

(1) We propose a revision of graph simulation [Milner 1989] (Section 2), referred to as *strong simulation*, by enforcing two conditions: (a) the duality to preserve the parent relationships and (b) the locality to eliminate excessive matches. For example, matching pattern graph Q_1 on data graph G_1 of Fig. 1 via strong simulation finds Bio_4 as the only match for Bio in Q_1.

(2) We identify a set of criteria for topology preservation, and show that strong simulation preserves the topology of pattern and data graphs (Section 3). We also prove that the number of matches via strong simulation is linear in the size of the data graph rather than exponential for subgraph isomorphism, and each match has a diameter bounded by the diameter of the pattern graph. Hence strong simulation indeed rectifies the problems of subgraph isomorphism and simulation. Moreover, we show that slight extensions to the notion make graph pattern matching intractable.

(3) We show that strong simulation retains the same complexity as earlier extensions of graph simulation [Fan et al. 2010a; Fan et al. 2011] by providing a *cubic*-time computation algorithm (Section 4). We also develop effective optimization techniques, notably a quadratic-time algorithm to minimize strong simulation queries.

[1]http:// www.linkedin.com

(4) We show that the locality of strong simulation allows us to develop a simple yet effective algorithm to find matches in distributed graphs (Section 5). To the best of our knowledge, this is among the first distributed algorithms for graph pattern matching, for which the need is evident when processing massive graphs (see *e.g.,* [Dean and Ghemawat 2004; Giatsoglou et al. 2011; Malewicz et al. 2010]).

(5) Using both real-life data (Amazon and YouTube) and synthetic data, we conduct an extensive experimental study (Section 6). We find that our algorithms for strong simulation scale well with large data graphs (*e.g.,* with 1.5×10^8 nodes). They are able to identify sensible matches that subgraph isomorphism fails to catch, and to eliminate excessive matches of graph simulation that do not make sense. We find that 70%-80% matches found by strong simulation are those found by subgraph isomorphism, while only 25%-38% for graph simulation. We also find that our optimization techniques are effective, reducing $1/4$ of running time on average.

We contend that strong simulation provides a promising model for graph pattern matching in emerging applications. Indeed, (1) in contrast to subgraph isomorphism, strong simulation is solvable in cubic-time rather than NP-complete, and moreover, due to its locality, it yields a set of matches with cardinality linear in the size of the data graph rather than exponential, where each match is bounded by the diameter of the pattern graph. (2) As opposed to graph simulation, it captures the topology of pattern graphs in its matches, such as parents, connectivity and cycles, by enforcing the duality and locality on matches, while it retains tractability as graph simulation. (3) Unlike graph simulation, the locality of strong simulation makes it possible to efficiently conduct graph pattern matching on distributed graphs. (4) As will be seen in Section 3, minor extensions to strong simulation would make graph pattern matching an intractable problem. In other words, strong simulation strikes a balance between the complexity and the capability to capture graph topology.

Organization. The rest of the paper is organized as follows. We introduce strong simulation in Section 2, and evaluates the notion analytically based on a set of criteria for topology preservation in Section 3. We provide a cubic-time algorithm for computing strong simulation in Section 4, followed by a distributed evaluation algorithm in Section 5. An experimental study is reported in Section 6, and related work is discussed in Section 7. Finally, Section 8 identifies open issues for future work.

2. STRONG SIMULATION

In this section, we first present basic notations of graphs. We then introduce the notion of strong simulation.

2.1. Preliminaries

We specify both data graphs and pattern graphs as follows.

Graphs. A *node-labeled directed graph* (or simply a *graph*) is defined as $G(V, E, l)$, where (1) V is a finite set of nodes; (2) $E \subseteq V \times V$ is a finite set of edges, in which (u, u') denotes an edge from nodes u to u'; and (3) l is a labeling function that maps each node u in V to a label $l(u)$ in a (possibly infinite) set Σ of labels.

Intuitively, the function $l()$ specifies node attributes, *e.g.,* keywords, blogs, comments, ratings, names, emails, companies [Amer-Yahia et al. 2007]; and the label set Σ denotes all such attributes. We simply denote G as (V, E) when it is clear from the context.

We next review some basic notations of graphs.

Subgraphs. Graph $H(V_s, E_s, l_H)$ is a *subgraph* of graph $G(V, E, l_G)$, denoted as $G[V_s, E_s]$, if (1) for each node $u \in V_s$, $u \in V$ and $l_H(u) = l_G(u)$, and (2) for each edge $e \in E_s$, $e \in E$. That is, subgraph $G[V_s, E_s]$ only contains a subset of nodes and a subset of edges of graph G.

Paths. A *directed* (resp. *undirected*) *path* ρ is a sequence of nodes $v_1, \ldots, v_n$ such that (v_i, v_{i+1}) (resp. either (v_i, v_{i+1}) or (v_{i+1}, v_i)) is an edge in G for $i \in [1, n-1]$. The *length* of ρ is the number of edges in ρ. Abusing notations for trees, we refer to v_{i+1} as a *child* of v_i (or v_i as a *parent* of v_{i+1}).

A *directed* (resp. *undirected*) *cycle* in a graph is a *directed* (resp. *undirected*) path with $v_1 = v_n$, having no repeated nodes other than the start and end nodes v_1 and v_n.

We say that a node is *reachable* from another node if and only if there exists an undirected path between them in the graph.

Connected components. A *connected component* of a graph is a subgraph in which any two nodes are connected to each other by undirected paths, and which is only connected to the nodes of itself, *i.e.,* a connected component is maximal. A graph that is itself connected has exactly one connected component, which is the entire graph.

Distance and diameter. Given two nodes v, v' in a graph G, the *distance* from v to v', denoted by $\mathsf{dist}(v, v')$, is the length of the shortest *undirected path* from v to v' in G.

The *diameter* of a connected graph G, denoted by d_G, is the longest shortest distance of all pairs of nodes in G, *i.e.,* $d_G = \mathsf{max}(\mathsf{dis}(v, v'))$ for all nodes v, v' in G.

We assume *w.l.o.g.* that pattern graphs are connected, as a common practice.

2.2. The Definition of Strong Simulation

We define strong simulation by enforcing two additional conditions on graph simulation [Milner 1989]: duality and locality. As will be seen in Sections 3 and 4, these conditions capture the topology of graphs and eliminate excessive matches to a large extent, while retaining a low PTIME computational complexity.

Consider a pattern graph $Q(V_q, E_q)$ and a data graph $G(V, E)$.

Graph simulation. To define strong simulation, we first review the notion of graph simulation [Milner 1989]. Data graph G matches pattern graph Q via *graph simulation*, denoted by $Q \prec G$, if there exists a binary *match relation* $S \subseteq V_q \times V$ such that:

(1) for each $(u, v) \in S$, u and v have the same label, *i.e.,* $l_Q(u) = l_G(v)$; and

(2) for each node u in V_q, there exists $v \in V$ such that (a) $(u, v) \in S$, and (b) for each edge $(u, u') \in E_q$, there exists an edge (v, v') in E such that $(u', v') \in S$.

Intuitively, graph simulation preserves the labels and the child relationship of a graph pattern in its match. It was initially proposed for the analyses of programs [Milner 1989], and studied for schema extraction from semi-structured data [Abiteboul et al. 1999]. Graph simulation and its extensions were recently employed for social networks [Brynielsson et al. 2010], and for graph pattern matching [Fan et al. 2010a; Fan et al. 2011] due to its low PTIME computational complexity [Henzinger et al. 1995].

Dual simulation. To capture graph topology, we first extend simulation by enforcing duality, to preserve the parent relationship as well.

Data graph G *matches* pattern graph Q via *dual simulation*, denoted by $Q \prec_D G$, if there exists a binary *match relation* $S \subseteq V_q \times V$ such that:

(1) for each $(u, v) \in S$, $l_Q(u) = l_G(v)$; and

(2) for each $u \in V_q$, there exists $v \in V$ such that (a) $(u, v) \in S$; (b) for each edge (u, u_1) in E_p, there is an edge (v, v_1) in E with $(u_1, v_1) \in S$; and, moreover, (c) for each edge (u_2, u) in E_q, there is an edge (v_2, v) in E with $(u_2, v_2) \in S$.

Intuitively, dual simulation enhances graph simulation by imposing an additional condition, to preserve both child and parent relationships (downward and upward mappings). Indeed, it is easy to verify that $Q \prec_D G$ if $Q \prec G$ with a binary *match relation* $S \subseteq V_q \times V$, and moreover, for each pair $(u, v) \in S$ and each edge (u_2, u) in E_q, there exists an edge (v_2, v) in E such that $(u_2, v_2) \in S$.

While there may be multiple matches in a graph G for a pattern Q, there exists a unique *maximum* match S_M in G for Q such that for any match S in G for P, $S \subseteq S_M$.

PROPOSITION 2.1. *For any pattern graph Q and data graph G with $Q \prec_D G$, there is a unique maximum match relation in G for Q.*

PROOF: (1) We first show that there exists a match relation. We consider all possible binary relations of $\{(u, v) \mid u$ is in Q, and v is in $G\}$, which satisfy conditions (1) and (2) of dual simulation. Note that those relations are not necessarily maximum, and the number of such possible relations is finite.

We define the maximum match relation to be a relation with the maximum number of elements, which, as will be seen shortly, is unique. Note that there must exist such a relation, as $Q \prec_D G$.

(2) We then show the uniqueness by contradiction. Assume that there exist two distinct maximum match relations S_1 and S_2. We then show that there exists a match relation S larger than both S_1 and S_2. Let $S = S_1 \cup S_2$. By the definition of dual simulation, one can readily verify that S is a match relation larger than both S_1 and S_2. This contradicts the assumption that both S_1 and S_2 are maximum.

By (1) and (2) above, we have the conclusion. □

Locality. We then introduce locality to capture more graph topology. To define the locality, we need the following notions.

Balls. For a node v in a graph G and a non-negative integer r, the *ball* with *center* v and *radius* r is a subgraph of G, denoted by $\hat{G}[v, r]$, such that (1) for all nodes v' in $\hat{G}[v, r]$, the shortest distance $\mathsf{dist}(v, v') \leq r$, and (2) it has exactly the edges that appear in G over the same node set.

We define the locality by requiring matches to be within a ball of a certain radius. Indeed, as observed in [Buchan and Croson 2004], when social distance increases, the closeness of relationships decreases and the relationships may become irrelevant. Hence it often suffices in practice to consider only those matches of a pattern graph that fall in a small ball.

To formalize this, we use the notion of match graphs, given as follows.

Match graphs. Consider a relation $S \subseteq V_q \times V$. The *match* graph *w.r.t.* S is a subgraph $G[V_s, E_s]$ of G, in which (1) a node $v \in V_s$ if and only if it is in S, and (2) an edge $(v, v') \in E_s$ if and only if there exists an edge (u, u') in Q with $(u, v) \in S$ and $(u', v') \in S$.

Intuitively, the match graph $G[V_s, E_s]$ *w.r.t.* S is the subgraph of G such that each of its nodes and edges plays a role in S.

Graph pattern matching via graph simulation or dual simulation is to find, when given any pattern graph Q and data graph G, the match graph *w.r.t.* the maximum match relation of Q and G if $Q \prec G$ or $Q \prec_D G$, and return empty otherwise.

We are now ready to define strong simulation.

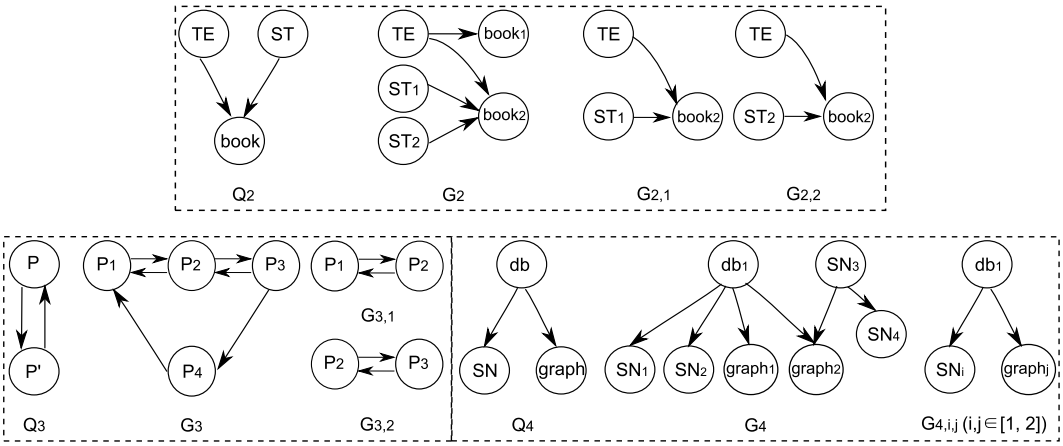


Fig. 2. Strong simulation

Strong simulation. Data graph G *matches* pattern graph Q via *strong simulation*, denoted by $Q \prec_D^L G$, if there exist a node v and a connected subgraph G_s in G that satisfy the following:

(1) $Q \prec_D G_s$ with the maximum match relation S;

(2) G_s is exactly the match graph *w.r.t.* S; and,

(3) G_s is contained in a ball $\hat{G}[v, d_Q]$ such that $v \in G_s$, where d_Q is the diameter of Q.

We refer to G_s as a *perfect* subgraph of G for Q.

Intuitively, a match G_s of pattern graph Q is required to satisfy the following conditions: (1) it preserves both the child and parent relationships of Q (condition 1 above); and (2) the nodes and edges needed to match Q are all contained in a ball with its radius determined by the diameter of Q (conditions 2 and 3); this rules out excessively large matches. As will be seen shortly, these conditions are justified for preserving graph topology and retaining low computational complexity.

Example 2.1: We first consider pattern graph Q_1 and data graph G_1 of Fig. 1. Observe the following.

(1) No subgraph of G_1 is isomorphic to Q_1. Indeed, there exist no directed cycles in G_1 that match the directed cycle $\mathsf{DM}, \mathsf{AI}, \mathsf{DM}$ in Q_1.

(2) When graph simulation is adopted, the entire data graph G_1 is included in the maximum match relation, which maps HR, SE, Bio, DM and AI in Q_1 to $\{\mathsf{HR}_1, \mathsf{HR}_2\}$, $\{\mathsf{SE}_1, \mathsf{SE}_2\}$, $\{\mathsf{Bio}_1, \mathsf{Bio}_2, \mathsf{Bio}_3, \mathsf{Bio}_4\}$, $\{\mathsf{DM}'_1, \mathsf{DM}'_2, \mathsf{DM}_1, \ldots, \mathsf{DM}_k\}$ and $\{\mathsf{AI}'_1, \mathsf{AI}'_2, \mathsf{AI}_1, \ldots, \mathsf{AI}_k\}$ in G_1, respectively.

(3) When it comes to strong simulation, the connected component G_c of G_1 that contains Bio_4 is the only match, which maps HR, SE, Bio, DM and AI in Q_1 to $\{\mathsf{HR}_2\}$, $\{\mathsf{SE}_2\}$, $\{\mathsf{Bio}_4\}$, $\{\mathsf{DM}'_1, \mathsf{DM}'_2\}$ and $\{\mathsf{AI}'_1, \mathsf{AI}'_2\}$ in G_1, respectively. Indeed, one can verify the following: (a) $Q_1 \prec_D G_c$, and in its match relation, Bio in Q_1 can only be mapped to Bio_4 in G_1; and (b) the ball with center Bio_4 and radius 3 (the diameter of Q_1) is exactly G_c. As opposed to graph simulation, the cycle $\mathsf{AI}_1, \mathsf{DM}_1, \ldots, \mathsf{AI}_k, \mathsf{DM}_k, \mathsf{AI}_1$ in G_1 is not part of the match. Indeed, this cycle is irrelevant and thus should be left out.

As another example, we consider pattern graphs Q_2, Q_3, Q_4 and data graphs G_2, G_3, G_4 shown in Fig. 2.

(4) Pattern Q_2 is to find a book recommended by both students (ST) and teachers (TE).

Table I. Summary of notations

Notations	Semantics
d_G	The diameter of a connected graph G
$G[V_s, E_s]$	A subgraph of G with node set V_s and edge set E_s
$\hat{G}[v, r]$	A ball in a graph G with center v and radius r
$Q \lhd G$	Data graph G matches pattern graph Q, via subgraph isomorphism
$Q \prec G$	Data graph G matches pattern graph Q, via graph simulation
$Q \prec_D G$	Data graph G matches pattern graph Q, via dual simulation
$Q \prec_D^L G$	Data graph G matches pattern graph Q, via strong simulation

When graph simulation is used, both book_1 and book_2 in G_2 are returned as matches, while book_1 is obviously not a good option. When strong simulation is adopted, book_2 is the only match by the *duality*, in a single match graph (union of $G_{2,1}, G_{2,2}$ in Fig. 2). When it comes to subgraph isomorphism, it returns two match graphs ($G_{2,1}, G_{2,2}$) instead of one, with book_2 as the match.

(5) Pattern Q_3 is to find people (P and P') who recommend each other. When graph simulation or dual simulation is used, all people (P_1, P_2, P_3 and P_4) in G_3 are found as matches, while P_4 is obviously not a good choice. When strong simulation is adopted, P_1, P_2 and P_3 are the only matches by the *locality*, in a single match graph (union of $G_{3,1}, G_{3,2}$ in Fig. 2). These are also the matches found via subgraph isomorphism, in two match graphs ($G_{3,1}, G_{3,2}$) instead of a single one.

(6) Pattern Q_4 is looking for papers on social networks (SN) cited by papers on databases (db), which in turn cite papers on graph theory (graph). When graph simulation is used, all papers on SN (SN_1, SN_2, SN_3 and SN_4) in G_4 are matches, while SN_3 and SN_4 are obviously excessive matches. When strong simulation is adopted, SN_1 and SN_2 are the only matches due to the *duality*, returned in a single match graph (union of $G_{4,i,j}$ with $i, j \in [1, 2]$ in Fig. 2). These are also the matches found by subgraph isomorphism, yet returned in four match graphs ($G_{4,i,j}$ for $i, j \in [1, 2]$) instead of one. □

Semantics. Strong simulation is to find, given any pattern graph Q and data graph G, the set of the *maximum* perfect subgraph G_s in each ball such that $Q \prec_D G_s$.

Here the size of a graph is the total number of its nodes and edges. One can verify the following, which assures that strong simulation is well defined.

PROPOSITION 2.2. *For any pattern graph Q and data graph G such that $Q \prec_D^L G$, there exists a unique set of maximum perfect subgraphs in G for Q.*

PROOF: It suffices to show that for each ball $\hat{G}[v, d_Q]$, there exists a unique maximum perfect subgraph containing the center node v for Q and $\hat{G}[v, d_Q]$ if there exists one.

We then show the uniqueness by contradiction. Assume that there exist two distinct perfect subgraphs $G_{s1}(V_{s1}, E_{s1})$ and $G_{s2}(V_{s2}, E_{s2})$ for Q and $\hat{G}[v, d_Q]$ that both contain the center node v . We then show that there exists a perfect subgraphs G_s larger than both G_{s1} and G_{s2}. Let $G_s = (V_{s1} \cup V_{s2}, E_{s1} \cup E_{s2})$. First, G_s is also a connected subgraph in $\hat{G}[v, d_Q]$ containing v. Second, by the definitions of dual simulation and strong simulation, one can readily verify that G_s is a perfect subgraph larger than both G_{s1} and G_{s2}. This contradicts the assumption that both G_{s1} and G_{s2} are maximum.

Putting these together, we have the conclusion. □

Remark. (1) Duality and locality are also imposed by subgraph isomorphism, but not by graph simulation. (2) One can readily extend strong simulation by supporting

bounds on the number of hops and regular expressions as edge constraints on pattern graphs, along the same lines as [Fan et al. 2010a; Fan et al. 2011].

We summarize notations used in Table I, in which we use $Q \lhd G$ to denote that Q matches G via subgraph isomorphism.

3. PROPERTIES OF STRONG SIMULATION

In this section, we first identify a set of criteria for topology preservation in graph pattern matching and for bounded match results. Based on the criteria, we then evaluate strong simulation, dual simulation, subgraph isomorphism and graph simulation. Finally, we explore possible extensions to strong simulation, and show that they lead to intractable problems.

Consider a connected pattern graph $Q = (V_q, E_q)$ and a data graph $G = (V, E)$.

3.1. Fundamental Properties

First, one can readily verify that subgraph isomorphism is a stronger notion than the other three, followed by strong simulation, dual simulation and graph simulation in this order. Intuitively, subgraph isomorphism preserves all topological structures between data graphs and pattern graphs.

PROPOSITION 3.1. *(1) If $Q \lhd G$, then $Q \prec^L_D G$; (2) if $Q \prec^L_D G$, then $Q \prec_D G$; and (3) if $Q \prec_D G$, then $Q \prec G$.*

PROOF: This can be easily verified by the definitions of subgraph isomorphism, strong simulation, dual simulation and graph simulation, which carry fewer restrictions one by one in this order. □

We next take a closer look at what structures are preserved by these matching notions, by giving a set of criteria.

(1) Children: if a node u in the pattern graph Q matches node v in the data graph G, then each child of u also matches a child of v.

All these notions preserve the child relationship.

(2) Parents: if a node u in the pattern graph Q matches node v in the data graph G, then each parent of u also matches a parent of v.

One can easily verify that subgraph isomorphism, strong simulation and dual simulation preserve the parent relationship, but graph simulation does not. A counterexample for graph simulation is given in Fig. 1.

(3) Weak connectivity: if the match graph in a data graph for a connected pattern graph is disconnected, then each connected component of the match graph matches the pattern graph. Unfortunately, graph simulation does not have this property. Consider the pattern graph Q_2 and data graph G_2 in Fig. 2, where the edge $(\mathsf{TE}, \mathsf{book}_2)$ is removed from G_2. Then while G_2 still matches Q_2 and G_2 is exactly the match graph *w.r.t.* the maximum match relation in G_2 for Q_2, there exists no connected component in G_2 that matches Q_2. Nevertheless, dual simulation, strong simulation and subgraph isomorphism all preserve weak connectivity, as stated below.

PROPOSITION 3.2. *If $Q \prec_D G$, then for any connected component G_c of the match graph* w.r.t. *the maximum match relation in G for Q, (1) $Q \prec_D G_c$, and (2) G_c is exactly the match graph* w.r.t. *the maximum match relation in G_c for Q.*

PROOF: Assume *w.l.o.g.* that G_c is a connected component of the match graph *w.r.t.* the maximum match relation $S \subseteq V_q \times V$ in G for Q.

(1) We first show that $Q \prec_D G_c$. Let u be any node in Q such that $(u, v) \in S$ and $v \in G_c$. Note that there must exist such a node u in Q. As G matches Q via dual simulation, for any neighboring (either child or parent) node x of u in Q, there must exist a neighboring node y of v such that $(x, y) \in S$. We then recursively consider the neighboring nodes of x, the neighboring nodes of the neighboring nodes of x, ..., until all nodes in Q are considered. Note that (a) the process above must terminate as pattern graph Q is connected, and (b) the process involves a set of nodes v in G and S such that there exists an undirected path from v to any of these nodes, which implies that all these nodes belong to G_c. Hence, $Q \prec_D G_c$, by the definition of dual simulation.

(2) We then show that the match graph *w.r.t.* the maximum match relation S_c in G_c for Q is exactly G_c. Let $S(G_c) = \{(u, v) \mid (u, v) \in S \text{ and } v \text{ in } G_c\}$. By (1) above, we know that $S(G_c)$ is a match relation in G_c for Q. Since G_c is the match graph *w.r.t.* $S(G_c)$, it suffices to show that $S_c = S(G_c)$. It is easy to see that $S(G_c) \subseteq S_c$, since otherwise S_c would not be the maximum match relation in G_c for Q. Thus we only need to show that $S_c \subseteq S(G_c)$. This holds since S would not be the maximum match relation in G for Q if $S_c \not\subseteq S(G_c)$. From these, we have $S_c = S(G_c)$, and hence, G_c is exactly the match graph *w.r.t.* the maximum match relation in G_c for Q. □

From Propositions 3.1 and 3.2, it follows that strong simulation and subgraph isomorphism also preserve the weak connectivity.

(4) Strong Connectivity: the match graph of a data graph for a connected pattern graph is always connected. This is a property of strong simulation and subgraph isomorphism, but not graph simulation or dual simulation. For example, consider the pattern graph Q_3 and data graph G_3 consisting of $G_{3,1}$ and $G_{3,2}$ in Fig. 2. The disconnected data graph G_3 is returned as the match graph in G_3 for Q_3 via either graph simulation or dual simulation. In contrast, the connected subgraphs $G_{3,1}$ and $G_{3,2}$ are returned as two match graphs in G_3 for Q_3 by strong simulation and subgraph isomorphism.

(5) Cycles: an undirected (resp. directed) cycle in Q must match an undirected (resp. directed) cycle in G. We show that graph simulation preserves directed cycles, and hence so do the other three matching notions.

PROPOSITION 3.3. *If $Q \prec G$ and there is a directed cycle in Q, then there must exist a matched directed cycle in the match graph* w.r.t. *any match relation in G for Q.*

PROOF: Assume *w.l.o.g.* that $\rho = u_1, u_2, \ldots, u_k, u_{k+1}$ is a directed cycle in Q such that (a) $u_1 = u_{k+1}$, and (b) there is a directed edge (u_i, u_{i+1}) in Q for each $i \in [1, k]$. Let S be any match relation in G for Q, and G_s be the match graph *w.r.t.* S. Moreover, for each u_i ($i \in [1, k]$) of path ρ in Q, let $S[u_i]$ be the set of nodes v in G_s such that $(u_i, v) \in S$. Also assume *w.l.o.g.* that $S[u_1] = \{v_{11}, \ldots, v_{1h}\}$.

(1) We first show that for each node $v_{1j} \in S[u_1]$ ($j \in [1, h]$), there exists a directed path to a node $v_{hj'} \in S[u_1]$. By the definition of graph simulation, for each node $v_i \in S[u_i]$ ($i \in [1, k]$), there exists a node $v_{i+1} \in S[u_{i+1}]$ with a directed edge (v_i, v_{i+1}) in G_s. Thus for each node $v_{1j} \in S[u_1]$ ($j \in [1, h]$), there exists a directed path to a node $v_{hj'} \in S[u_{k+1}]$. That is, there exists a directed path to a node $v_{1j'} \in S[u_1]$ for each node $v_{1j} \in S[u_1]$ ($j \in [1, h]$). Recall that $u_1 = u_{k+1}$ and $S[u_k] = S[u_{k+1}]$.

(2) We next show that there exists a directed cycle in G_s that matches the directed cycle ρ in Q. (a) We first construct h directed paths as follows: $\rho_1 = v_{11}, \ldots, v_{1j_1}$, $\rho_2 = v_{1j_1}, \ldots, v_{1j_2}$, ..., and $\rho_h = v_{1j_{h-1}}, \ldots, v_{1j_h}$ such that nodes $v_{11}, v_{1j_1}, \ldots, v_{1j_{h-1}}, v_{1j_h} \in S[v_1]$. Note that the analysis in (1) implies that there must exist h such directed paths. (b) We then construct a cycle from these h directed paths. By connecting the h paths

one by one in the order, we get a longer directed path ρ, which contains at least $h+1$ nodes in $S[u_1]$. Since there are only h distinct nodes in $S[u_1]$, there must exist a node $v_{1i} \in S[u_1]$ that appears at least twice in ρ, from which one can easily derive a cycle.

From these, we have the conclusion. □

However, as shown in Example 1.1, graph simulation may match an undirected cycle in a pattern graph with a tree in a data graph. In contrast, dual simulation (as well as subgraph isomorphism and strong simulation) preserves undirected cycles.

PROPOSITION 3.4. *If $Q \prec_D G$ and there is an undirected cycle in Q, then there must exist a matched undirected cycle in the match graph* w.r.t. *any match relation in G for Q.*

PROOF: This is verified along the same lines as Proposition 3.3. The only difference is that here we can readily construct a set of undirected paths, from which we can derive an undirected cycle, by the definition of dual simulation that considers both parent and child relationships. □

(6) Locality: the diameter of a matched subgraph in G must be bounded by a function in the diameter of the pattern graph. This allows us to check a match locally, by inspecting a subgraph with a bounded diameter only.

Strong simulation has the locality property, and so does subgraph isomorphism. In contrast, neither graph simulation nor dual simulation has the locality property (see Examples 1.1 and 2.1).

PROPOSITION 3.5. *If $Q \prec^L_D G$, then for all perfect subgraphs G_s of G, the diameter of G_s is bounded by $2 * d_Q$, where d_Q is the diameter of Q.*

PROOF: This follows from the definition of strong simulation since balls are connected graphs whose diameters are less or equal than d_Q. For any two nodes u, u' in a perfect subgraph G_s in a ball $\hat{G}(v, d_Q)$, $\mathsf{dist}(u, u') \leq \mathsf{dist}(u, v) + \mathsf{dist}(v, u') \leq 2 * d_Q$. Hence, the diameter of G_s is bounded by $2 * d_Q$. □

(7) Bounded Matches: there should be a bounded number of matches, and each match is small enough to inspect. As remarked earlier, subgraph isomorphism may yield exponentially many matched subgraphs. While graph simulation and dual simulation return a single match relation, the relation is often too large to inspect. In contrast, strong simulation strikes a balance: the number of matches is bounded by the number of nodes in the data graph, and each matched subgraph has a bounded diameter determined by the pattern graph only (Proposition 3.5).

PROPOSITION 3.6. *The number of maximum perfect subgraphs of G is bounded by the number of nodes in G.*

PROOF: By the definition of strong simulation, there exists at most one maximum perfect subgraph for each ball $\hat{G}(v, d_Q)$, where v is a node in data graph G, and d_Q is the diameter of pattern Q. The number of balls is bounded by the number of nodes in G, and so is the number of maximum perfect subgraphs. □

These results are summarized in Table II. They tell us that strong simulation preserves much more topological structures between pattern graphs and data graphs than graph simulation, and moreover, possesses the locality property.

Table II. Topology preservation and bounded matches

Property	Matching			
	Graph simulation	Dual simulation	Strong simulation	Subgraph isomorphism
Children	✓	✓	✓	✓
Parents	×	✓	✓	✓
Weak Connectivity	×	✓	✓	✓
Strong Connectivity	×	×	✓	✓
Directed cycles	✓	✓	✓	✓
Undirected cycles	×	✓	✓	✓
Locality	×	×	✓	✓
Bounded Matches	✓	×	✓	×
Bisimilarity	×	×	×	✓
Bounded cycles	×	×	×	✓

3.2. In Search for Tractable Boundary in Matching

One might want to find a notion of graph pattern matching that preserves maximum graph topology, and characterize PTIME along the same lines as how Fagin's theorem characterizes NP [Papadimitriou 1994]. This is, however, very challenging. Indeed, as observed in [Grohe 2010], in graph theory Fagin's theorem implies that "if no logic captures PTIME, then PTIME $\neq$ NP".

Below we present two negative results: extending strong simulation makes its computation jump from PTIME to NP-hard.

Bounded cycles. Given a pattern graph Q and a data graph G such that $Q \prec G$ with the maximum match relation S, the *bounded cycle problem* (BCP) is to determine whether the longest cycle in the match graph *w.r.t.* S is bounded by the longest one in Q. Obviously bounded cycles are a desirable locality property that one would have wanted to further impose on strong simulation. Unfortunately, this additional condition would make graph pattern matching intractable.

THEOREM 3.7. *The bounded cycle problem is co*NP*-hard even when pattern graphs contain a single cycle.*

PROOF: We show that the BCP problem is coNP-hard even when the pattern graph Q is a single cycle graph, by reduction from the longest cycle problem (LCP) to the complement of the BCP problem. The LCP problem is to determine whether the length of the longest cycle in a graph is greater than or equal to k. It is known to be NP-complete (cf. [Papadimitriou 1994]).

Given an instance (*i.e.,* a graph G_l and an integer k) of the LCP problem, we construct an instance (*i.e.,* a pattern graph Q and a data graph G_b) of the BCP problem, such that G_l has a cycle with at least k nodes if and only if a cycle in Q matches a cycle in G_b not bounded by the length of the longest cycle in Q.

More specifically, we construct the instance of the BCP problem as follows:

(1) The pattern graph Q is simply a cycle with $k-1$ nodes, in which all nodes have the same label, and moreover,

(2) the data graph G_b is derived from G_l, by labeling all nodes in G_l with the same label as the nodes of Q.

Observe that the matched cycle in G_b is not bounded by $k-1$, the length of the longest cycle in Q, if and only if G_l contains a cycle with at least k nodes. Therefore, the BCP problem is coNP-hard. □

Bisimilarity. One might be tempted to use graph bisimulation [Milner 1989] rather than graph simulation in graph pattern matching. A data graph G *matches* a pattern graph Q via *graph bisimulation*, denoted by $Q \sim G$, if $Q \prec G$ with the maximum match relation S and $G \prec Q$ with the inverse S^- of S as its maximum match relation. Graph pattern matching via graph bisimulation is to find all subgraphs G_s of a data graph G such that $Q \sim G_s$. Clearly graph bisimulation preserves more topological structures than graph simulation. Indeed, it is a notion stronger than graph simulation but weaker than subgraph isomorphism.

However, graph pattern matching via graph bisimulation becomes intractable. Indeed, subgraph bisimulation is NP-hard [Dovier and Piazza 2003], although graph bisimulation is solvable in PTIME [Milner 1989]. In contrast, subgraph simulation is equivalent to graph simulation, *i.e.,* checking whether there exists a subgraph G_s of G such that $Q \prec G_s$ is the same as checking whether $Q \prec G$.

4. AN ALGORITHM FOR STRONG SIMULATION

In this section, we show that graph pattern matching via strong simulation retains the same complexity as earlier extensions [Fan et al. 2010a; Fan et al. 2011] of graph simulation, while it is able to preserve graph topology better.

The main results of this section are as follows.

THEOREM 4.1. *For any pattern graph Q and data graph G, it can be done in cubic time to determine whether $Q \prec^L_D G$, and to find the set of maximum perfect subgraphs of G* w.r.t. *Q.*

THEOREM 4.2. *For any pattern graph Q with diameter d_Q, it can be done in quadratic time to find a minimum pattern graph Q_m such that Q_m and Q find the same result on any data graph by using d_Q as the radius of balls, via strong simulation.*

We first prove Theorem 4.1 by providing a cubic-time algorithm for computing strong simulation. We then show Theorem 4.2 by proposing optimization techniques.

4.1. A Cubic-time Algorithm

Algorithm. The algorithm, referred to as Match, is shown in Fig. 3. Given a pattern graph Q and a data graph G, it returns the set of maximum perfect subgraphs G_s by inspecting those balls of radius d_Q centered at each node of G.

To present algorithm Match, we first describe its procedures.

Procedure DualSim. It takes as input a pattern graph $Q(V_q, E_q)$ and a ball $\hat{G}[w, d_Q]$ with center w and radius d_Q, and finds the maximum match relation S_w in $\hat{G}[w, d_Q]$ for Q.

For each node u in Q, the set $\mathsf{sim}(u)$ contains candidate nodes in the ball, initially all its nodes with the same label as u (lines 1–2). By the definition of dual simulation, a node v is removed from $\mathsf{sim}(u)$ unless (1) if there is a parent node u' of u, then there exists a parent node $v' \in \mathsf{sim}(u')$; and (2) if there is a child node u' of u, then there exists a child node $v' \in \mathsf{sim}(u')$. Hence, to preserve the child relationships, if $(u, u') \in E_q$ and $v \in \mathsf{sim}(u)$, but there exist no nodes $v' \in \mathsf{sim}(u')$ such that (v, v') is an edge in $\hat{G}[w, d_Q]$, then v cannot be matched to u, and hence is removed from $\mathsf{sim}(u)$ (lines 4–6). Similarly, to preserve the parent relationships, if $(u', u) \in E_q$ and $v \in \mathsf{sim}(u)$, but there exist no nodes $v' \in \mathsf{sim}(u')$ such that (v', v) is an edge in $\hat{G}[w, d_Q]$, then v cannot be matched to u, and hence is removed from $\mathsf{sim}(u)$ (lines 7–9). This process is repeated until there are no more changes (lines 3–9). Finally, S_w is returned (lines 10–12).

Algorithm Match(Q, G)

Input: Pattern graph Q with diameter d_Q and data graph $G(V, E)$.
Output: The set Θ of maximum perfect subgraphs of G for Q.

1. $\Theta := \emptyset$;
2. **for each** ball $\hat{G}[w, d_Q]$ in G **do**
3. $S_w :=$ DualSim$(Q, \hat{G}[w, d_Q])$;
4. $G_s :=$ ExtractMaxPG$(Q, \hat{G}[w, d_Q], S_w)$;
5. **if** $G_s \neq$ *nil* **then**
6. $\Theta := \Theta \cup \{G_s\}$;
7. **return** Θ.

Procedure DualSim$(Q, \hat{G}[w, d_Q])$

Input: Pattern graph $Q(V_q, E_q)$ and ball $\hat{G}[w, d_Q]$.
Output: The maximum match relation S_w in $\hat{G}[w, d_Q]$ for Q.

1. **for each** $u \in V_q$ in Q **do**
2. sim$(u) :=$ $\{v \mid v$ is in $\hat{G}[w, d_Q]$ **and** $l_Q(u) = l_G(v)\}$;
3. **while** there are changes **do**
4. **for each** edge (u, u') in E_Q and **each** node $v \in$ sim(u) **do**
5. **if** there is no edge (v, v') in $\hat{G}[w, d_Q]$ with $v' \in$ sim(u') **then**
6. sim$(u) :=$ sim$(u) \setminus \{v\}$;
7. **for each** edge (u', u) in E_Q and **each** node $v \in$ sim(u) **do**
8. **if** there is no edge (v', v) in $\hat{G}[w, d_Q]$ with $v' \in$ sim(u') **then**
9. sim$(u) :=$ sim$(u) \setminus \{v\}$;
10. **if** sim$(u) = \emptyset$ **then return** $\emptyset$;
11. $S_w :=$ $\{(u, v) \mid u \in V_q, v \in$ sim$(u)\}$;
12. **return** S_w.

Procedure ExtractMaxPG$(Q, \hat{G}[w, d_Q], S_w)$

Input: Pattern Q, ball $\hat{G}[w, d_Q]$ with maximum match relation S_w.
Output: The maximum perfect subgraph G_s in $\hat{G}[w, d_Q]$ for Q if any.
1. **if** w does not appear in S_w **then**
2. **return** *nil*;
3. Construct the matching graph G_m *w.r.t.* S_w;
4. **return** the connected component G_s containing w in G_m.

Fig. 3. Algorithm Match for strong simulation

Procedure ExtractMaxPG. It takes as input a pattern graph Q, a ball $\hat{G}[w, d_Q]$, and the maximum match relation S_w, and finds the maximum perfect subgraph G_s in the ball if there is one. By Proposition 3.2, the procedure simply finds the connected component containing w in the match graph *w.r.t.* S_w after constructing the match graph *w.r.t.* S_w.

Algorithm Match. We are now ready to present Match. For each node w in the data graph G, (1) it computes the maximum match relation S_w of Q and the ball $\hat{G}[w, d_Q]$ by invoking DualSim (line 2); (2) it finds the perfect subgraph G_s in $\hat{G}[w, d_Q]$ via ExtractMaxPG (line 3); and (3) G_s is added to the set Θ if it exists (line 4). After all balls in G are checked, it returns the set Θ of maximum perfect subgraphs (line 5).

Example 4.1: Consider pattern graph Q_1 ($d_{Q_1} = 3$) and the ball with center Bio_4 and radius = 3 in data graph G_1 of Fig 1. Note that the ball is exactly the connected component G_c with node Bio_4 in G_1. We show how Algorithm Match works on Q_1 and G_c. Initially, HR, SE, Bio, AI and DM in Q_1 match $\{\mathsf{HR}_2\}$, $\{\mathsf{SE}_2\}$, $\{\mathsf{Bio}_4,\}$, $\{\mathsf{AI}'_1, \mathsf{AI}'_2\}$ and $\{\mathsf{DM}'_1,$

Input: Pattern graph $Q = (V_q, E_q, l_Q)$.
Output: A minimized equivalent pattern graph Q_m of Q.

1. Compute the maximum match relation S in Q for Q via dual simulation;
2. Compute equivalence classes of nodes in Q *w.r.t.* S;
3. For each equivalence class eq, create a node for Q_m with the same label as the nodes in eq;
4. Connect equivalence classes with necessary edges in Q_m;
5. **return** Q_m.

Fig. 4. Algorithm minQ for minimizing pattern graphs

$\mathsf{DM}'_2\}$ in G_c, respectively (lines 1–2, DualSim). The algorithm finds no nodes to be removed from $\mathsf{sim}(\mathsf{u})$ for all nodes u in Q_1 in this case (lines 3–10, DualSim). Hence Match returns G_c as the maximum perfect subgraph in the ball (line 6, Match). □

Correctness & Complexity. The correctness of algorithm Match is assured by the following. (1) There is at most one maximum perfect subgraph in each ball of G (Proposition 2.2). (2) Procedure ExtractMaxPG returns the maximum perfect graph in ball $\hat{G}[v, d_Q]$, by Proposition 3.2. (3) The correctness of procedure DualSim can be verified along the same lines as its counterpart for graph simulation [Henzinger et al. 1995], by further dealing with parent relationships.

By using the BFS method [Diestel 2005], it takes procedure BuildBall (not shown here) $O(|V| + |E|)$ time to build a ball $\hat{G}[w, d_Q]$. For each ball, procedure ExtractMaxPG finds its maximum perfect subgraph in $O(|V|)$ time since finding pairwise disconnected components is linear-time equivalent to finding strongly connected components, which is in linear time [Cormen et al. 2001]. By leveraging the algorithm developed in [Henzinger et al. 1995], procedure DualSim can be done in $O((|V_q| + |E_q|)(|V| + |E|))$ time. Thus algorithm Match is in $O(|V|(|V| + (|V_q| + |E_q|)(|V| + |E|)))$ time.

Algorithm Match is in cubic time, and this completes the proof of Theorem 4.1.

4.2. Optimization Techniques

We next present optimization techniques for algorithm Match, by means of query minimization, dual simulation filtering and connectivity pruning.

4.2.1. Query Minimization. We first explore query minimization, which is important for any query language [Abiteboul et al. 1995].

We say that two pattern graphs Q and Q' are *equivalent*, denoted by $Q \equiv Q'$, if and only if they find the same result on any data graph. We also say that a pattern graph Q is *minimum* if it has the least size $|Q|$, *i.e.,* the total number of nodes and edges, among all equivalent pattern graphs.

Theorem 4.2 follows from Lemmas 4.3 and 4.4 given below.

LEMMA 4.3. *When fixing the radius of balls, if two pattern graphs are equivalent via dual simulation, then they are equivalent via strong simulation.*

PROOF: Assume that $Q_1 \equiv Q_2$ via dual simulation. By the definition of strong simulation, it is trivial to know that $Q_1 \equiv Q_2$ via strong simulation. □

LEMMA 4.4. *For any pattern graph Q, (1) there exists a unique (up to isomorphism) minimum equivalent pattern graph, via dual simulation, that finds the same maximum match relation as Q on any data graph; and (2) there exists a quadratic time algorithm to find its minimum equivalent pattern graph.*

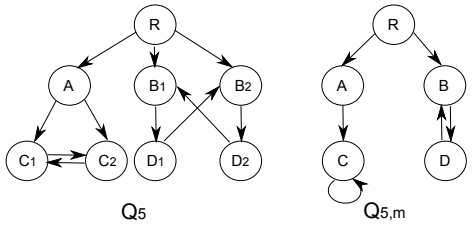


Fig. 5. Example for query minimization

Leveraging these, Algorithm Match can be improved as follows. Given pattern graph Q, we first compute its minimum equivalent pattern graph Q_m, and then we compute strong simulation *w.r.t.* Q_m and diameter d_Q.

Algorithm. As a proof of Lemma 4.4, we present algorithm minQ for minimizing pattern graphs, shown in Fig. 4. It takes as input a pattern graph Q, and returns a minimum equivalent pattern graph Q_m of Q, via dual simulation.

For any pattern graph Q, it first computes the maximum match relation S by treating Q as both a pattern graph and a data graph (line 1). It then computes *equivalence classes* for nodes in Q such that nodes u and v are in the same class if and only if both $(u, v) \in S$ and $(v, u) \in S$ (line 2). Finally, it constructs the minimum equivalent pattern graph Q_m as follows (lines 3–4). (a) For each equivalence class eq, it creates a node eq in Q_m, and (b) there is an edge (eq, eq′) in Q_m if and only if there exist nodes $u \in$ eq and $u' \in$ eq′ such that there exists an edge (u, u') in Q.

Example 4.2: Taking as input the pattern graph Q_5 given in Fig. 5, where nodes carry labels, and nodes with the same label further use subscripts to indicate the distinction.

Algorithm minQ works as follows.

(1) It first computes the maximum match relation S of Q_5 and Q_5, via dual simulation, yielding S = {(R, R), (B_i, B_j), (C_i, C_j), (D_i, D_j)} ($i, j \in [1, 2]$).

(2) It then computes five equivalence classes: $\mathsf{eq}_R = \{R\}$, $\mathsf{eq}_A = \{A\}$, $\mathsf{eq}_B = \{B_1, B_2\}$, $\mathsf{eq}_C = \{C_1, C_2\}$, and $\mathsf{eq}_D = \{D_1, D_2\}$.

(3) Finally, it constructs the minimum pattern graph $Q_{5,m}$ of Q_5, shown in Fig. 5: (a) For each equivalence class eq_x, where $x \in \{R, A, B, C, D\}$, it creates a node labeled with x; and (b) it creates an edge from node x to y in $Q_{5,m}$ if and only if there exist nodes $u \in \mathsf{eq}_x$ and $v \in \mathsf{eq}_y$ such that (u, v) is an edge in Q_5. □

Remark. Observe that for all the nodes in the same equivalence class, algorithm Match conducts essentially the same computation on them. Hence minimized pattern graphs reduce redundant computation. This is also confirmed by the complexity analysis of algorithm Match, which tells us that the smaller pattern graphs are, the better the algorithm performs. The technique is effective on (1) pattern graphs in which multiple nodes are equivalent and can thus be reduced, and on (2) data graphs in which the number of nodes that match equivalent pattern nodes is large.

Correctness. The correctness of algorithm minQ is assured by the following.

(1) For any data graph G, the maximum match relation S in G for Q is always the same as the maximum match relation S_m in G for Q_m. Hence, $Q \equiv Q_m$.

(2) $|Q_m| \leq |Q'|$ for any Q' such that $Q' \equiv Q$.

(3) For any two minimum equivalent pattern graphs Q_m and Q'_m, there is a bijective function from Q_m to Q'_m such that (a) for any node u in Q_m, $f(u)$ is a node in Q'_m with

the same label, and (b) (u, v) is an edge in Q_m if and only if $(f(u), f(v))$ is an edge in Q'_m, *i.e.,* Q_m and Q'_m are equivalent up to isomorphism.

We show that algorithm minQ satisfies these conditions as follows.

(I) We first show that algorithm minQ satisfies condition (1) above, *i.e.,* $Q \equiv Q_m$.

To show that $Q \equiv Q_m$, it suffices to show that $Q \prec_D Q_m$ and $Q_m \prec_D Q$.

(i) We first show that $Q \prec_D Q_m$. Let $S = \{(u, \mathsf{eq}) \mid u \text{ in } Q \text{ and } \mathsf{eq} \text{ in } Q_m\}$ such that $u \in \mathsf{eq}$, *i.e.,* node u belongs to the equivalence class eq. We next show that S is a match relation in Q_m for Q, from which we conclude $Q \prec_D Q_m$. Indeed, for any node u in Q, (a) there exists node eq_u in Q_m such that u and eq_u have the same label and $(u, \mathsf{eq}_u) \in S$, (b) for each edge (u, v) in Q, there is an edge $(\mathsf{eq}_u, \mathsf{eq}_v)$ in Q_m, where $(v, \mathsf{eq}_v) \in S$, and (c) for each edge (w, u) in Q, there is an edge $(\mathsf{eq}_w, \mathsf{eq}_u)$ in Q_m, where $(w, \mathsf{eq}_w) \in S$. From these it follows that S is a match relation in Q_m for Q, via dual simulation.

(ii) We then show that $Q_m \prec_D Q$. Let $S^{-1} = \{(\mathsf{eq}_u, u) \mid (u, \mathsf{eq}_u) \in S\}$. As argued in (i) above, we can show that $Q_m \prec Q$ with S^{-1} as a match relation in Q for Q_m.

By (i) and (ii), we have $Q \equiv Q_m$.

(II) We next show that algorithm minQ satisfies condition (2) above, by contradiction.

Assume first that Q' is a pattern graph such that (a) $Q' \equiv Q$, (b) $|Q'| < |Q_m|$, and (c) given Q' as input, algorithm minQ outputs Q' (otherwise, we simply replace Q' with the one generated by minQ). We then show $|Q'| \geq |Q_m|$, which contradicts our assumption.

Let EQ_m and EQ' be the two sets of equivalence classes for Q_m and Q', respectively, produced by algorithm minQ. Then each node in Q_m or Q' forms a separate equivalence class that contains itself only. To show that $|Q'| \geq |Q_m|$, it suffices to show that there exists a total function f from EQ_m to EQ' such that for any $\mathsf{eq}_1, \mathsf{eq}_2 \in \mathsf{EQ}_m$, there is an edge $(\mathsf{eq}_1, \mathsf{eq}_2)$ in Q_m if and only if $(f(\mathsf{eq}_1), f(\mathsf{eq}_2))$ is an edge in Q'. Let S_1 and S_2 be the maximum match relations in Q_m for Q' and in Q' for Q_m, respectively.

We next construct such a mapping function f, by letting $(\mathsf{eq}, \mathsf{eq}') \in f$ if and only if both $(\mathsf{eq}, \mathsf{eq}') \in S_1$ and $(\mathsf{eq}', \mathsf{eq}) \in S_2$.

(i) We first show that f is indeed a function by proving (a) f is total; and (b) for any $\mathsf{eq} \in \mathsf{EQ}_m$, there exists exactly one equivalence class eq' $\in \mathsf{EQ}'$ such that $f(\mathsf{eq}) = \mathsf{eq}'$.

(a) We first prove that mapping f is a function by contradiction.

Assume first that there is an equivalence class eq in EQ_m such that there exist two distinct equivalence classes eq'_1 and eq'_2 in EQ', satisfying that $(\mathsf{eq}, \mathsf{eq}'_1) \in f$ and $(\mathsf{eq}, \mathsf{eq}'_2) \in f$. We next show that $\mathsf{eq}'_1 = \mathsf{eq}'_2$, which contradicts our assumption. Let S_m be the maximum match relation in Q_m for Q_m. Since $(\mathsf{eq}, \mathsf{eq}'_1) \in f$, we have $(\mathsf{eq}, \mathsf{eq}'_1) \in S_1$ and $(\mathsf{eq}'_1, \mathsf{eq}) \in S_2$. Similarly, since $(\mathsf{eq}, \mathsf{eq}'_2) \in f$, we have $(\mathsf{eq}, \mathsf{eq}'_2) \in S_1$ and $(\mathsf{eq}'_2, \mathsf{eq}) \in S_2$. By the definition of dual simulation, one can easily verify that both $(\mathsf{eq}'_1, \mathsf{eq}'_2) \in S_m$ and $(\mathsf{eq}'_2, \mathsf{eq}'_1) \in S_m$. This tells us that eq'_1 and eq'_2 are equivalent via dual simulation, and hence $\mathsf{eq}'_1 = \mathsf{eq}'_2$, a contradiction to our previous assumption.

(b) We then show that function f is total by contradiction.

Assume first that there exists an $\mathsf{eq} \in \mathsf{EQ}_m$ such that there exists no $\mathsf{eq}' \in \mathsf{EQ}'$ that satisfies both $(\mathsf{eq}, \mathsf{eq}') \in S_1$ and $(\mathsf{eq}', \mathsf{eq}) \in S_2$. Since $Q' \equiv Q_m$, there are two cases for eq and eq': (1) $(\mathsf{eq}, \mathsf{eq}') \in S_1$, but not $(\mathsf{eq}', \mathsf{eq}) \in S_2$, or (2) $(\mathsf{eq}, \mathsf{eq}') \in S_2$, but not $(\mathsf{eq}', \mathsf{eq}) \in S_1$.
(1) If $(\mathsf{eq}, \mathsf{eq}') \in S_1$, but not $(\mathsf{eq}', \mathsf{eq}) \in S_2$, then there must exist eq_2 in EQ_m such that $(\mathsf{eq}', \mathsf{eq}_2) \in S_2$ since $Q' \equiv Q_m$. Now we have $(\mathsf{eq}, \mathsf{eq}') \in S_1$ and $(\mathsf{eq}_2, \mathsf{eq}') \in S_2$. By the definition of dual simulation, one can easily verify that both eq and eq_2 are equivalent via dual simulation, and hence $\mathsf{eq} = \mathsf{eq}_2$, a contradiction to our previous assumption.
(2) Similarly, one can verify the case when $(\mathsf{eq}, \mathsf{eq}') \in S_1$, but not $(\mathsf{eq}', \mathsf{eq}) \in S_2$.

By (a) and (b) above, we conclude that mapping f is indeed a total function.

Input: Pattern graph Q, match relation S *w.r.t.* $Q \prec_D G$ and ball $\hat{G}[w, d_Q]$.
Output: The maximum perfect subgraph in $\hat{G}[w, d_Q]$ for Q.

1. S_w := project S onto $\hat{G}[w, d_Q]$;
2. *filterSet* := $\emptyset$;
3. **for each** $(u, v) \in S_w$ such that v is a border node **do**
4. **if** there is no child of v in $\mathsf{sim}(u_1)$ such that $(u, u_1) \in E_q$ **then**
5. *filterSet*.push(u, v);
6. **if** there is no parent of v in $\mathsf{sim}(u_2)$ such that $(u_2, u) \in E_q$ **then**
7. *filterSet*.push(u, v);
8. **while** (*filterSet* $\neq \emptyset$) **do**
9. (u, v) := *filterSet*.pop();
10. $S_w := S_w \setminus \{(u, v)\}$;
11. **for each** (u, u_1) in Q **do**
12. **for each** child v_1 of v in $\mathsf{sim}(u_1)$**do**
13. **if** there is no parent of v_1 in $\mathsf{sim}(u)$ **then**
14. *filterSet*.push(u_1, v_1);
15. **for each** (u_2, u) in Q **do**
16. **for each** parent v_2 of v in $\mathsf{sim}(u_2)$**do**
17. **if** there is no child of v_2 in $\mathsf{sim}(u)$ **then**
18. *filterSet*.push(u_2, v_2);
19. **if** there exists u in Q such that $\mathsf{sim}(u) = \emptyset$ **then**
20. $S_w := \emptyset$;
21. **return** ExtractMaxPG (Q, $\hat{G}[w, d_Q]$, S_w)

Fig. 6. Algorithm dualFilter for dual simulation filtering

(ii) We then show that for any $\mathsf{eq}_1, \mathsf{eq}_2 \in \mathsf{EQ}_m$, there is an edge ($\mathsf{eq}_1$, eq_2) in Q_m if and only if ($f(\mathsf{eq}_1)$, $f(\mathsf{eq}_2)$) is an edge in Q', by contradiction.

Assume first that (eq_1, eq_2) in Q_m, but not ($f(\mathsf{eq}_1)$, $f(\mathsf{eq}_2)$) in Q'. Then by how algorithm minQ constructs edges, and the definition of dual simulation, $(\mathsf{eq}_1, f(\mathsf{eq}_1))$ does not belong to f, a contradiction to the assumption. Conversely, it is similar for the case that ($f(\mathsf{eq}_1)$, $f(\mathsf{eq}_2)$) in Q', but not (eq_1, eq_2) in Q_m.

Putting these together, we have shown that $|Q_m| \leq |Q'|$ for any Q' such that $Q' \equiv Q$.

(III) We finally show that algorithm minQ satisfies condition (3) above.

The function f constructed above is bijective. This can be verified by proving that the inverse f^- of f is a total and injective function, via a similar argument to (2).

Putting (II) and (III) together, we have that f is a total, surjective and injective function. That is, f is a bijection from the nodes of Q_m to the nodes of Q'. Therefore, Q' and Q_m have the same number of nodes and edges, and are isomorphic to each other.

Complexity. Algorithm minQ is in $O((|V_q| + |E_q|)^2)$ time. Indeed, steps (1), (2) and (3) of minQ take $O((|V_q| + |E_q|)^2)$ time, $O(|V_q|^2)$ time and $O(|E_q|)$ time, respectively.

This completes the proofs of Lemma 4.4 and Theorem 4.2.

4.2.2. Dual Simulation Filtering. Our second optimization technique aims to avoid redundant checking of balls in the data graph. Most algorithms of graph simulation (*e.g.,* [Henzinger et al. 1995]) recursively refine the match relation by identifying and removing false matches. As observed in [Fan et al. 2010a], it is much easier to deal with node or edge deletions than their insertions. In light of this, we compute the maximum match relation for dual simulation first, and then project the match relation on each ball to compute strong simulation.

We say that a node w in a ball $\hat{G}[v, d_Q]$ is a *border node* if $\mathsf{dist}(v, w) = d_Q$. We also refer to those nodes reachable from a border node as its *affected* nodes.

With these, one can easily verify the following:

Observation. Initially, only the border nodes in a ball are possible to be removed from the candidates $\mathsf{sim}(u)$ of any node u in a pattern graph, *i.e.,* invalid matches.

This suggests an order to process nodes in $\hat{G}[v, d_Q]$: starting from its border nodes, we inspect affected nodes only following an increasing order based on their distances from border nodes. This minimizes unnecessary computation. Note that the border nodes can be marked when constructing balls. Hence this incurs little extra complexity.

Algorithm. To do this, we first compute the maximum match relation S, via dual simulation, over the entire data graph G by invoking procedure DualSim in Fig. 3. We then project S onto each ball. When computing the maximum match relation on a ball, we simply start with the *border* nodes, and identify invalid matches using algorithm dualFilter shown in Fig. 6, a revised version of DualSim.

We next present algorithm dualFilter. It takes as input a pattern graph Q, the maximum match relation S in G for Q that is found via dual simulation, and a ball $\hat{G}[w, d_Q]$. It returns the maximum perfect subgraph in $\hat{G}[w, d_Q]$ for Q. More specifically, dualFilter first projects the maximum match relation S onto ball $\hat{G}[w, d_Q]$, yielding relation S_w (line 1). It then iteratively marks and removes those invalid matches stored in a queue *filterSet* (lines 2–18), initially empty (line 2). To do this, it first inspects those matches in S_w that contain a border node, to find invalid matches (lines 4–7). The invalid matches found are stored in *filterSet* (lines 5 and 7). It then processes those marked invalid matches one by one (lines 6–15). Each invalid match (u, v) with affected node v is removed from S_w (line 10). The relation S_w is then processed along the same lines as procedure DualSim (lines 11–18), but following the order of invalid matches in *filterSet*. Finally, the algorithm extracts the perfect subgraph by invoking procedure ExtractMaxPG, and returns the subgraph (line 21).

Example 4.3: We illustrate how the filtering technique improves the performance of algorithm Match by considering the pattern graph Q_6 with diameter $d_{Q_6} = 3$ and data graph G_6 given in Fig. 7. The maximum match relation S in G_6 for Q_6 via dual simulation is the set of matches (node pairs): $\{(A, A_2), (B, B_2), (A', A_3), (B', B_3), (C, C)\}$. That is, $\mathsf{sim}(A) = \{A_2\}$, $\mathsf{sim}(B) = \{B_2\}$, $\mathsf{sim}(A') = \{A_3\}$, $\mathsf{sim}(B') = \{B_3\}$ and $\mathsf{sim}(C) = \{C\}$.

The filtering method then projects the match relation S on each ball, and checks the results. It finds the following:

(1) There exist invalid matches in two balls: $\hat{G}_6[A_1, 3]$ and $\hat{G}_6[B_1, 3]$, by inspecting their border nodes. For ball $\hat{G}_6[A_1, 3]$, after projecting S on $\hat{G}_6[A_1, 3]$, we get $S_w = \{(A, A_2), (B, B_2)\}$. Here B_2 is a border node of $\hat{G}_6[A_1, 3]$. Starting with B_2, dualFilter finds that both (A, A_2) and (B, B_2) are invalid matches. Similarly for ball $\hat{G}_6[B_1, 3]$.

(2) In contrast, there exist no invalid matches in the other balls: $\hat{G}_6[A_2, 3]$, $\hat{G}_6[B_2, 3]$, $\hat{G}_6[A_3, 3]$, $\hat{G}_6[B_3, 3]$ and $\hat{G}_6[C, 3]$. This is found by inspecting border nodes in each ball. Hence the final match relation in any of these balls is exactly the same as the initial projected match relation of S on the ball.

As a result, only two balls $\hat{G}_6[A_1, 3]$ and $\hat{G}_6[B_1, 3]$ are necessarily processed by algorithm dualFiler, while no more processing is needed for the other five balls. That is, the filtering method prunes unnecessary processing and speeds up algorithm Match. □

Remark. Observe the following. (1) The match result of procedure Dualsim is also used as the initial match set $\mathsf{sim}(u)$ for each node u in Q (line 1, procedure dualFilter). Therefore, procedure Dualsim incurs little extra cost. (2) Procedure Dualsim filters those nodes

in a data graph G that do not match any nodes in a pattern graph Q. Pattern graphs are often small, and hence a large portion of nodes in G may not match any pattern node in Q, when they either have a label that does appear in Q, or when they do not satisfy the matching condition of dual simulation. Procedure Dualsim catches such nodes and filters a large portion of nodes in data graphs. (3) If a node v in G does not match any node in Q, then there is no need to consider the ball centered at v at all, and hence the number of balls (line 2, algorithm Match) is often reduced. From these we can see that the optimization technique is effective when a large number of nodes in a data graph G do not find a match in a pattern graph Q, as commonly found in practice.

Correctness. We next show the correctness of dualFilter. Let S_1 and S_2 be the S_w at line 1 and line 21 of algorithm dualFilter, respectively. Let S_3 be the maximum match relation in ball $\hat{G}[w, d_Q]$ for Q, returned by procedure DualSim. Observe the following:

(i) $S_3 \subseteq S_1$ and $S_2 \subseteq S_1$;

(ii) for any $(u, v) \in S_1$, if v is not a border node, then for any parent or child of u in Q, there exists a parent or child v' of v such that $(u', v') \in S_1$; and

(iii) for any $(u, v) \notin S_2$, its removal is due to the removal of matches at line 1 of algorithm dualFilter. Here a match (u', v') is affected by the removal of (u, v) if (a) u' is a parent (resp. child) of u and (b) v is the *only* child (resp. parent) of v' that matches u.

To show the correctness of dualFilter, it suffices to show that that $S_2 = S_3$, by showing $S_2 \subseteq S_3$ and $S_3 \subseteq S_2$. It is obvious that $S_3 \subseteq S_2$ since $S_3 \subseteq S_1$ and for any pair (u, v) removed from S_1 in dualFilter, (u, v) is not in S_3. Hence we only need to show that $S_2 \subseteq S_3$. We next show that for any $(u, v) \in S_1$, if $(u, v) \notin S_3$, then $(u, v) \notin S_2$, by contradiction, from which we conclude that $S_2 \subseteq S_3$.

Assume first that there exists $(u, v) \notin S_3$, but $(u, v) \in S_2$. Then there exists no parent (or child) v' of v such that $(u', v') \in S_3$, where u' is a parent (or child) of u in Q.
(1) If v is a border node, then (u, v) is not in S_2 by the definition of dual simulation, a contradiction to our previous assumption.
(2) Otherwise, if v is not a border node, we then recursively consider those parents (resp. children) u^1 of u and parents (resp. children) v^1 of v in Q such that there exists (u^1, v^1) in S_1, but $(u^1, v^1) \notin S_3$:

- If v^1 is a border node, then (u^1, v^1) is not in S_2 by the definition of dual simulation.
- Otherwise, if v^1 is not a border node, we then repeat the process (2).

Due to the observation (ii), this process will finally terminate when all nodes involved are border nodes. As a result, this leads to $(u, v) \notin S_2$, a contradiction.

From these the correctness of algorithm dualFilter follows.

Complexity. For its complexity, observe that it takes $O(|V|(|V|+|E|))$ time to construct all balls, and $O((|V_q|+|E_q|)(|V|+|E|))$ time to compute the maximum match relation in G for Q via dual simulation, along the same lines as algorithm Match. For each ball $\hat{G}[w, d_Q]$, it takes at most $O((|V_q|+|E_q|)(|V_{\hat{G}[w,d_Q]}|+|E_{\hat{G}[w,d_Q]}|))$ time. Putting these together, dualFilter takes $O(|V|(|V|+(|V_q|+|E_q|)(|V|+|E|)))$ time in total. Although the worst case complexity is the same as the complexity of Match (shown in Fig. 3), as demonstrated by the example and as will be shown by our experimental study, the optimization technique is indeed effective in practice.

4.2.3. Connectivity pruning. Proposition 3.2 tells us that in a ball $\hat{G}[v, r]$, only the connected component containing the ball center v needs to be considered. Hence, those nodes not reachable from v can be pruned early. Our last main optimization technique does precisely this. It reduces the search space for checking dual-simulation, and can be easily incorporated into Algorithm Match, as illustrated below.

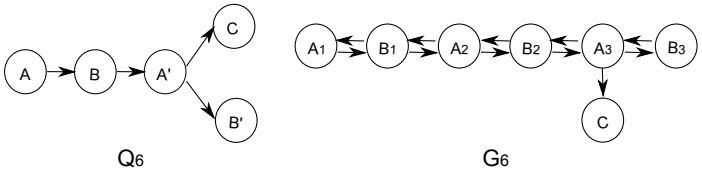


Fig. 7. Example for dual simulation filtering

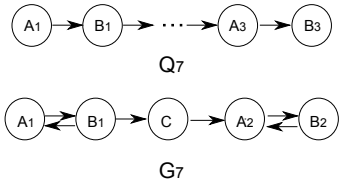


Fig. 8. Example for connectivity pruning

Example 4.4: Consider pattern graph Q_7 and data graph G_7 shown in Fig. 8, in which diameters $d_{Q_7} = 5$ and $d_{G_7} = 4$. Here $d_{Q_7} > d_{G_7}$, so a ball with any center node of G_7 is exactly G_7 itself. When conducting dual simulation of Q_7 on ball $\hat{G}_7[A_1, 5]$, for instance, the pruning method first finds an initial sim(v) set for each node v in Q_7, by mapping A_i in Q_7 to A_j in $\hat{G}_7[A_1, 5]$ ($i \in [1, 3]$, $j \in [1, 2]$). This yields two connected components in $\hat{G}_7[A_1, 5]$: CC_1 containing nodes $\{A_1, B_1\}$ and CC_2 containing $\{A_2, B_2\}$, in which only CC_1 contains the center node A_1 (recall the notion of connected graphs from Section 2). By Proposition 3.2, the pruning method safely removes all those nodes that are not in CC_1 from sim(u), for any node $u \in Q_7$, without affecting the final matches. That is, it prunes invalid matches early and thus improves the performance of Match. □

Putting things together. We next show how to integrate those optimization techniques into Match, yielding a version of algorithm Match that supports all optimization strategies, referred to as Match$^+$.

Given a pattern graph Q and a data graph G, algorithm Match$^+$ works as follows:

Step 1: It first invokes algorithm minQ to get a minimized Q_m for pattern graph Q.

Step 2: It then removes all nodes in data graph G with labels not appeared in Q_m, yielding a set of connected components of G.

Step 3: It conducts procedure DualSim on each connected component of G.

Step 4: It then uses connectivity pruning on each ball of each connected component of G to further reduce the projected match relation returned at Step 3 on each ball.

Step 5: It finally utilizes algorithm dualFilter on each ball of each connected component of G with the refined projected match relation returned at Step 4, and returns all the perfect subgraphs.

The correctness of Match$^+$ is asserted by Proposition 3.2 (for Steps 2, 3) and the correctness of connectivity pruning (for Step 4) and dual simulation filtering (for Step 5).

As will be seen in Section 6, Match$^+$ significantly outperforms Match.

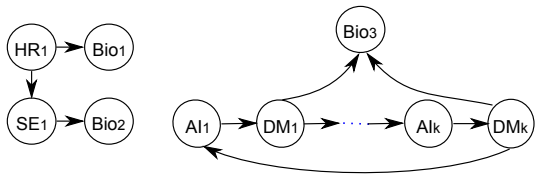


Fig. 9. Data graph G_s for Example 5.1

5. COMPUTING STRONG SIMULATION ON DISTRIBUTED GRAPHS

In this section, we study distributed evaluation of strong simulation. We first provide an algorithm based on data locality for strong simulation on distributed graphs (Section 5.1) and then propose optimization techniques for it (Section 5.2).

5.1. An Algorithm for Strong Simulation on Distributed Graphs

When evaluating a query on a large dataset, one wants to partition the data and distribute its fragments to multiple machines, such that the query can be evaluated in parallel, as advocated by, *e.g.,* MapReduce [Dean and Ghemawat 2004]. Moreover, it is common to find real-life datasets already partitioned and distributed. For instance, to find the complete information of a person, one may have to query several social networks (*e.g.,* Facebook, Picassa and Youtube) to collect her data. These highlight the need for developing distributed algorithms for evaluating graph queries. However, as observed in [Malewicz et al. 2010], graph algorithms often exhibit poor data locality and hence, may incur prohibitive overhead on network traffic.

We next show that strong simulation demonstrates data locality and hence, allows efficient distributed evaluation.

Data locality. Consider a graph G that is partitioned into $(G_1, \ldots, G_k)$ such that each G_i is stored in site M_i ($i \in [1, k]$). We want to evaluate a pattern graph Q on G, while minimizing unnecessary data shipment from one site to another. This is, however, rather challenging when pattern matching is defined in terms of graph simulation.

Example 5.1: Consider again the pattern graph Q_1 and given in Fig. 1, and the data graph G_s shown in Fig. 9, which is the subgraph of G_1 by removing the connected component with Bio_4 from G_1. Suppose that G_s is fragmented and distributed. Then to decide whether $Q_1 \prec G_s$, we have to ship all subgraphs of G_s to a single site to re-assemble G_s. Indeed, (1) the match graph of Q_1 and G_s via graph simulation is the entire G_s; and (2) removing any node or edge from G_s makes $Q_1 \not\prec G_s$. This tells us that it is hard to conduct graph simulation in the distributed setting without incurring high network traffic. □

In contrast, we show that strong simulation has the data locality. Indeed, strong simulation can be computed in the distributed setting, guaranteeing that the total data shipment is bounded by the set of balls $\hat{G}[v_b, d_Q]$ in G such that v_b is in some G_i but it has a direct neighbor node not in G_i. We refer to G_i as a *fragment* of G and v_b as a *boundary* node of fragment G_i.

Algorithm. To verify the data locality of strong simulation, we provide a distributed algorithm for graph pattern matching via strong simulation, denoted by dMatch and shown in Fig. 10. Given a pattern graph Q and a partitioned data graph $G = \{G_1, \ldots, G_k\}$ with G_i placed in site M_i ($i \in [1, k]$), it returns the set of maximum perfect subgraphs in G for Q, by invoking two distributed processes as follows.

Input: Pattern graph Q with diameter d_Q and partitioned data graph $G = \{G_1, \ldots, G_k\}$ with G_i placed in site M_i ($i \in [1, k]$).
Output: The set Θ of maximum perfect subgraphs in G for Q.

Coordinator.
1. **send** pattern graph Q to all sites M_i ($i \in [1, k]$);
2. $\Theta := \emptyset$; $S := \emptyset$;
3. **upon** receiving (M_i, Θ_i):
4. $\Theta := \Theta \cup \Theta_i$; $S := S \cup \{M_i\}$;
5. **if** sizeof$(S) = k$ **then return** Θ.

Site M_i.
1. **upon** receiving pattern graph Q:
2. Trigger necessary data shipments for G_i;
3. **after** receiving all data shipments (subgraphs of G) from other sites:
4. $G_i :=$ Merge G_i with the received subgraphs of G;
5. $\Theta_i :=$ Match(Q, G_i);
6. **send** (M_i, Θ_i) to the coordinator.

Fig. 10. Algorithm dMatch for strong simulation on distributed graphs

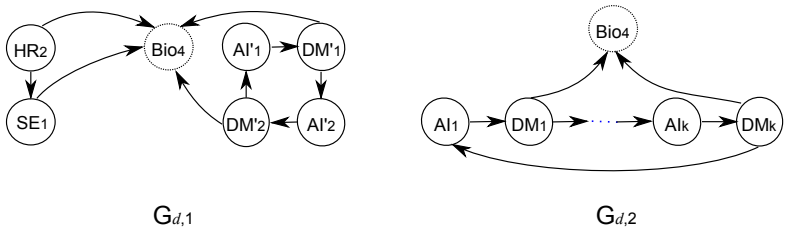


Fig. 11. Data graph G_d for Example 5.2

Coordinator. When a site, referred to as the coordinator and denoted by M_Q, receives a pattern graph Q, it sends the same Q to each site M_i for $i \in [1, k]$ (line 1). It then monitors messages sent back from all those sites (line 3), and assembles their partial results via union (line 4). When partial results are returned from all the sites, it returns the final result, *i.e.,* the set of maximum perfect subgraphs in G for Q (line 5).

Site M_i. When a site M_i receives Q (line 1), it first finds those balls $\hat{G}[v_b, d_Q]$, where v_b is a boundary node in G_i. It then sends $\hat{G}[v_b, d_Q]$ to all those sites M_j in which v_b has direct neighbor (parent or child) nodes in M_j (line 2). After receiving all the data shipments from other sites (line 3), it updates G_i by incorporating the set of shipped balls (line 4), and invokes the centralized algorithm Match to compute the matches in G_i for Q (*i.e.,* a set of maximum perfect subgraphs of G_i for Q) as a partial result Θ_i in G for Q (line 5). It finally sends Θ_i back to the coordinator (line 6).

Example 5.2: Consider the pattern graph Q_1 with diameter $d_{Q_1} = 3$ in Fig. 1 and the data G_d shown in Fig. 11 which is derived from G_1 in Fig. 1 as follows. Let G'_1 be a derived graph of G_1 by dropping Bio_3 and its corresponding edges, and connecting DM_1 and DM_k *only* to Bio_4, and let G_d be the connected component of G'_1 containing Bio_4. Graph G_d is partitioned into two fragments $G_{d,1}$ and $G_{d,2}$ that contain nodes $\{\mathsf{Bio}_4, \mathsf{AI}_1, \mathsf{DM}_1, \ldots, \mathsf{AI}_k, \mathsf{DM}_k\}$ and $\{\mathsf{HR}_2, \mathsf{SE}_2, \mathsf{Bio}_4, \mathsf{AI}'_1, \mathsf{DM}'_1, \mathsf{AI}'_2, \mathsf{DM}'_2\}$, respectively, and fragments $G_{d,1}$ and $G_{d,2}$ are placed at sites M_1 and M_2, respectively. Observe that node Bio_4 is the only boundary node for both fragments.

We next show how algorithm dMatch works on pattern graph Q_1 and data graph G_d. The coordinator M_1 first sends pattern graph Q_1 to all sites. After site M_2 (resp. M_1) receives Q_1, it triggers the data shipments of balls. In this case, (a) M_2 retrieves the ball with center node Bio_4 in $G_{d,1}$ from M_1, and (b) M_1 retrieves the ball with center node Bio_4 in $G_{d,2}$ from M_2. Then sites M_1 and M_2 compute in parallel the maximum perfect graphs for their local subgraphs by calling the centralized algorithm Match. In this case, there is a single maximum perfect graph in sites M_1 and M_2, which is exactly $G_{d,2}$ itself. Finally, the coordinator M_1 collects and returns the unique maximum perfect graph G_d in G_d for Q_1. □

The correctness of dMatch is asserted by the data locality property of strong simulation, with the bound on network traffic mentioned above. Indeed, for any boundary node v_b of fragment G_i, dMatch collects ball $\hat{G}[v_b, d_Q]$ in site M_i, and hence does not miss any valid match. Furthermore, the distributed computation strategy is generic: it is applicable to any data graph G *regardless of* how G is partitioned and distributed.

Remark. To maintain a complete view of data graphs G, in each site we maintain locally how a fragment is connected to other fragments located in other sites. That is, for all boundary nodes of a fragment, all their edges in G are stored locally, including those with endpoints across fragments located in distinct sites, a typical setting for evaluating distributed queries (see *e.g.,* [Cong et al. 2007; Ma et al. 2012; Fan et al. 2012b]).

5.2. Optimization Techniques

We next present optimization techniques for our distributed algorithm dMatch, by means of *ball pruning* and the optimized algorithm Match^+ for local computations. We consider a pattern graph $Q(V_q, E_q)$ and a partitioned data graph $G = \{G_1, \ldots, G_k\}$ such that fragment $G_i(V_i, E_i)$ is placed in site M_i ($i \in [1, k]$).

Ball pruning aims to reduce unnecessary shipments of balls, and it is based on a notion of *partial match relations*, which is introduced as follows.

Partial match relations. We say that a binary relation $S_i \subseteq V_q \times V_i$ is a *partial match relation* via dual simulation in fragment G_i for pattern graph Q if for each $(u, v) \in S_i$,

(1) nodes u and v have the same label; and

(2) for each parent (resp. child) u' of u in Q, there exists a parent (resp. child) v' of v in G_i such that (a) u' and v' simply have the same label if v is a boundary node of G_i, or (b) $(u', v') \in S_i$, otherwise.

The difference between a partial match relation and a match relation via dual simulation given in Section 2 is that the former also deals with boundary nodes. When there are no boundary nodes involved, these two notions are equivalent.

Remark. (1) The union of all partial match relations S_i in G_i for Q is a super set of the maximum match relation S in the entire data graph G for Q, *i.e.,* $S \subseteq \bigcup_{i=1}^{k} S_i$.

(2) A slightly revised version of procedure DualSim (Section 4), which further deals with boundary nodes, can be adopted to compute the *maximum partial match relation* S_i in G_i for Q, via dual simulation.

With these, one can easily verify the following properties. Consider a ball $\hat{G}[v_i, d_Q]$ located in site M_i, in which node v_i is a boundary node of fragment G_i.

(1) Ball $\hat{G}[v_i, d_Q]$ is necessarily shipped to another site M_j if and only if there is a boundary node v_j in fragment G_j such that there exists an edge (v_i, v_j) or (v_j, v_i) in the match graph *w.r.t.* the maximum match relation S in G for Q.

(2) If node v_i does not appear in the maximum partial match S_i, then there is no need to ship ball $\hat{G}[v_i, d_Q]$ to any other site.

(3) If there exists no parent or child of node v_i that appears in the maximum partial match S_j in another fragment G_j for Q, then there is no need to ship ball $\hat{G}[v_i, d_Q]$ to site M_j. Recall that we maintain locally all the neighboring nodes for boundary nodes.

Here properties (2) and (3) follow from property (1) and the fact $S \subseteq \bigcup_{i=1}^{k} S_i$.

Algorithm. We are now ready to introduce the ball pruning based algorithm.

(1) After receiving the pattern graph Q from the coordinator, all sites M_i ($i \in [1, k]$) compute, *in parallel*, the maximum partial match relation S_i via dual simulation in fragment G_i for Q, by calling the revised procedure DualSim.

(2) Then all sites M_i send in parallel those boundary nodes v_i of fragment G_i to all other sites M_j such that node v_i (a) appears in S_i and (b) has a parent or child node in fragment G_j located in site M_j.

(3) After that, all sites M_i trigger in parallel the shipments of balls $\hat{G}[v_i, d_Q]$ to another site M_j if boundary node v_i (a) appears in S_i and (b) has a parent or child node appearing in S_j in fragment G_j for Q.

(4) After ball shipments are done, all sites M_i call the optimized algorithm Match$^+$*in parallel* to compute the local match result Θ_i, by treating the maximum partial match S_i as the initial candidate matches.

(5) Finally, all partial match results are sent back and assembled at the coordinator.

We refer to this optimized version of Algorithm dMatch as dMatch$^+$, by (a) adopting Match$^+$, instead of Match, for local computation in dMatch; and (b) leveraging the ball pruning technique for data shipments.

Example 5.3: Consider the pattern graph Q_1 and the distributed data graph G_d discussed in Example 5.2. We show how the optimized algorithm dMatch$^+$ works on Q_1 and G_d and prunes unnecessary data shipments.

After receiving pattern graph Q_1 for both sites M_1 and M_2, they first call dMatch$^+$ to compute in parallel the partial match relations in fragments $G_{d,1}$ and $G_{d,2}$ in sites M_1 and M_2, respectively. Here the partial match relation is *empty* in site M_1, and hence, no ball shipments are needed from sites M_1 to M_2. Only site M_1 retrieves the ball with center Bio$_4$ from M_2. Finally, dMatch$^+$ computes the maximum perfect graphs following the same line as dMatch in Example 5.2.

Note that here dMatch$^+$ avoids ball shipments with ball pruning. □

Remark. Along the same lines as the dual simulation filtering technique for the centralized algorithm dMatch, a large number of nodes in data graphs are filtered after computing the maximum partial match relation. As a result, unnecessary ball shipments are avoided. In addition, we only need to ship those balls whose centers are attached to the nodes falling into a maximum partial match relation, and hence further reduce ball shipments. This optimization technique is particularly effective when the number of boundary nodes is large. As will be seen in Section 6, dMatch$^+$ outperforms dMatch in terms of data shipment, especially when Q is relatively large.

6. EXPERIMENTAL STUDY

We next present an extensive experimental study of strong simulation. Using both real-life social networks and synthetic data, we conducted three sets of experiments to

evaluate: (1) the effectiveness of strong simulation vs. conventional subgraph isomorphism [Ullmann 1976] and graph simulation [Henzinger et al. 1995], (2) the performance of our centralized algorithm Match, and (3) the performance of our distributed algorithm dMatch. We also evaluated the effectiveness of our optimization techniques..

Experimental setting. We used the following datasets.

Real-life graph data. We used two real-life network datasets.

(1) Amazon records a product co-purchasing network with 548,552 product nodes and 1,788,725 product-product directed edges[2]. An edge from product x to y indicates that if people buy x, then the chances are that they will also buy y with high probability.

(2) YouTube provides a video network with 155,513 video nodes and 3,110,120 video-video directed edges[3]. An edge from video x to y indicates that if one watches x, then he is also very likely to watch y.

Synthetic graph generator. We adopted the graph-tool library[4] to produce both pattern graphs and data graphs. It is controlled by three parameters: the number n of nodes, the number n^α of edges, and the total number l of node labels. Given n, α, and l, the generator produces a graph with n nodes, n^α edges, and the nodes are randomly labeled from a set of l labels.

Algorithms. We implemented the following algorithms, all in Python.

(1) Our centralized algorithms Match and Match$^+$.

(2) Our distributed algorithms dMatch and dMatch$^+$.

(3) The centralized algorithm [Henzinger et al. 1995], denoted by Sim, and the distributed algorithm [Ma et al. 2012], denoted by dSim, for graph simulation.

(4) The approximate matching algorithm TALE of [Tian and Patel 2008].

(5) The approximate matching algorithm, denoted by MCS, that utilizes the approximation algorithm of [Kann 1992] for computing maximum common subgraphs.

(6) We adopted the VF2 algorithm [Cordella et al. 2004] for subgraph isomorphism in the iGraph package [Csardi and Nepusz 2006]. We also implemented a distributed algorithm based on VF2 for subgraph isomorphism, denoted by dVF2, by replacing Match with VF2 in algorithm dMatch for strong simulation. Note that subgraph isomorphism is a stronger notion than strong simulation, and hence, it also has the data locality property. This guarantees the correctness of the distributed algorithm dVF2.

Consider pattern graph $Q(V_q, E_q)$ and data graph $G(V, E)$. For approximate matching algorithms TALE and MCS, there are possibly $2^{|V|}$ number of subgraphs of G to compare with Q, beyond reach in practice. Hence, we chose to compare the subgraphs of G having the same number of nodes as Q. We adopted the same setting as [Tian and Patel 2008] for TALE here. For MCS, a subgraph $G_s(V_s, E_s)$ of G matches pattern graph Q if $\frac{|\mathsf{mcs}(Q,G_s)|}{\max(|V_q|,|V_s|)} \geq 0.7$, where $|\mathsf{mcs}(Q, G_s)|$ is the number of nodes in the maximum common subgraph $\mathsf{mcs}(Q, G_s)$ of Q and G_s computed via the algorithm of [Kann 1992].

[2]http://snap.stanford.edu/data/index.html
[3]http://netsg.cs.sfu.ca/youtubedata/
[4]http://projects.skewed.de/graph-tool/

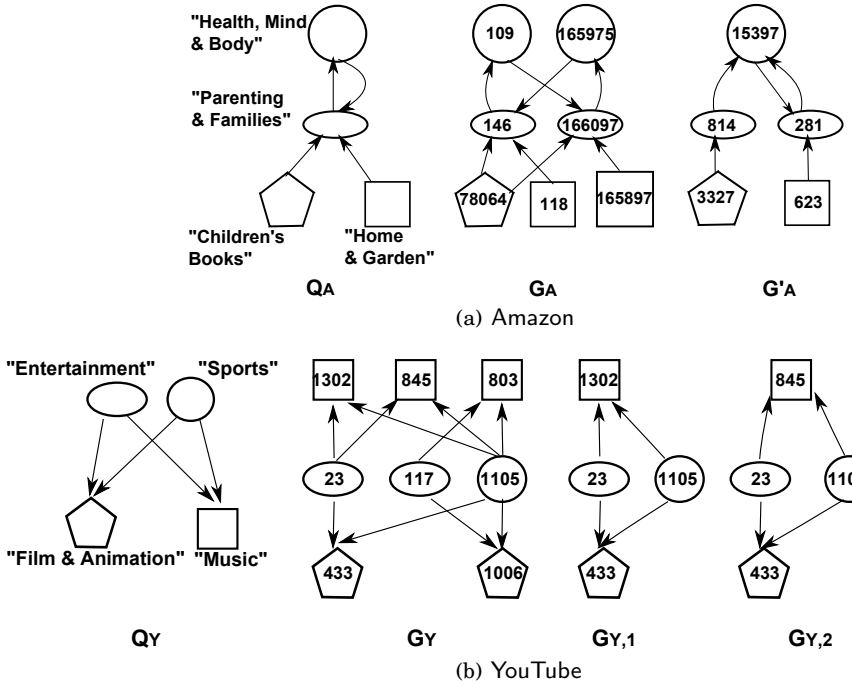


Fig. 12. Real-life matches on real-life data

All the experiments were run on a cluster of 30 machines, all with Intel(R) Xeon(R) E5620 CPU and 48GB of memory. Each test was repeated over 5 times, and the average is reported here.

Experimental results. We next present our findings. In all the experiments, we fixed $l = 200$, and set $\alpha = 1.2$ by default when generating pattern and data graphs.

Exp-1: Quality of matches. In the first set of experiments, we evaluated the quality of matches found by strong simulation vs. matches found by subgraph isomorphism and graph simulation. We first illustrate two example matches of strong simulation on real-life data. We then test the quality of matches with five measures: three *closeness* measures, the *number* of matched subgraphs and the *sizes* of matched subgraphs. These together are to show that strong simulation is capable of capturing structures of pattern and data graphs.

(1) We first designed pattern graphs, and manually checked the quality of matches returned by Match, VF2 and Sim. We find that Match is able to identify sensible matches.

We illustrate this with two example pattern graphs. Two real-life pattern graphs Q_A and Q_Y are shown in Figures 12(a) and 12(b), respectively.

(1) Pattern graph Q_A is to find all "Parenting & Families" books in the Amazon network data (a) that are co-purchased with both "Children's Books" and "Home & Garden" books; and (b) that are co-purchased with "Health, Mind & Body" books and vice versa.

(2) Pattern graph Q_Y poses a request on the YouTube network data, searching for all "Entertainment" videos (a) that are related to both "Film & Animation" and "Music" videos; and further, (b) for each such "Entertainment" video x, there is another "sports"

video that is related to the "Film & Animation" and "Music" videos to which the video x is related.

In data graphs G_A and G_Y, nodes are books and videos, respectively, labeled with their ids, and they only match the nodes of Q_A and Q_Y with the same geometry shapes, *e.g.,* circles, ellipses, and regular squares and pentagons.

The match results of Q_A and Q_Y are shown in Figures 12(a) and 12(b), respectively.

For pattern graph Q_A, subgraph G_A is a sensible match found by Match, but it was not found by VF2. Subgraph G'_A is a match found by Sim in which the "Parenting & Families" books are not co-purchased with both "Children's Books" and "Home & Garden" books, among other things, and was successfully filtered by Match and VF2. These tell us that strong simulation is able to identify sensible matches that subgraph isomorphism fails to catch, and moreover, to eliminate excessive matches found by graph simulation that do not make sense.

For pattern graph Q_Y, subgraph G_Y is a match found by Match, while subgraphs $G_{Y,1}$, $G_{Y,2}$ and $G_{Y,3}$ are three separate matches found by VF2. This example shows how strong simulation reduces the sizes of matches found by subgraph isomorphism, without loss of information.

(2) To further measure the quality of matches found, we use the following three *closeness* measures:

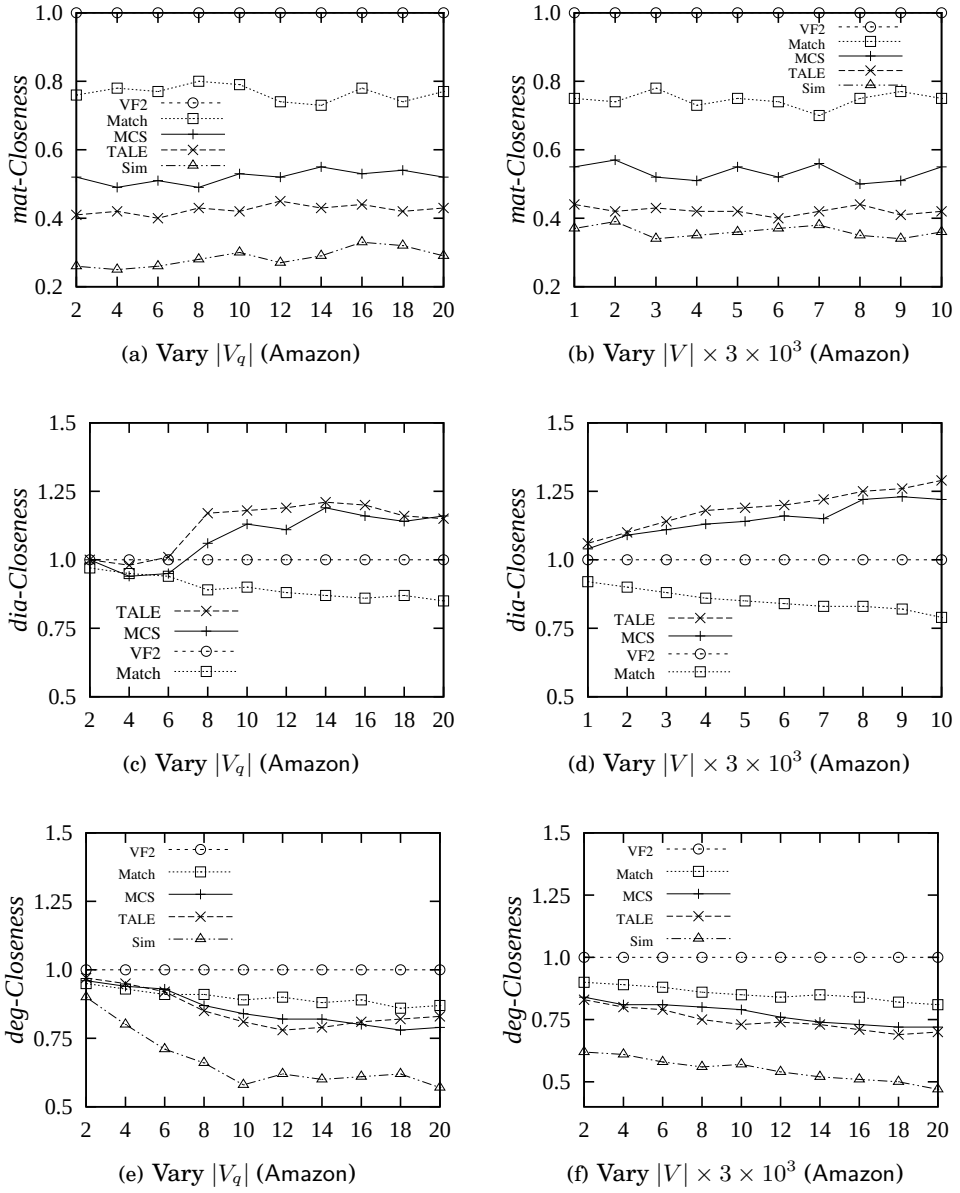
mat–closeness = #matches_subIso / #matches_found,
dia–closeness = diameter(pattern_graphs) / average–diameter(matched_subgraphs_found),
deg–closeness = average–degree(pattern_graphs) / average–degree(matched_subgraphs_found).

Here (a) #matches_subIso and #matches_found are the total numbers of nodes in matches found by VF2 and those by a comparative algorithm (Sim, Match, VF2, TALE, MCS), respectively; (b) diameter(pattern_graphs) and average–diameter(matched_subgraphs_found) are the diameters of the pattern graphs and the average diameters of matched subgraphs returned by those algorithms, respectively; and (c) average–degree(pattern_graphs) and average–degree(matched_subgraphs_found) are the average degrees of pattern graphs, and the matched subgraphs returned by the algorithms, respectively. Recall that matches found by VF2 are also matches found by Match and Sim, by Proposition 3.1. Hence mat–closeness is essentially the ratio of matched nodes found by VF2 to the entire matches found by Sim, Match, VF2, TALE or MCS. Intuitively, mat–closeness, dia–closeness and deg–closeness are to evaluate the ability of those graph pattern matching algorithms to identify sensible node matches, and to preserve the diameter and average degree of pattern graphs. Note that for all these closeness measures, the closer to 1, the better, and for VF2, its mat–closeness, dia–closeness and deg–closeness are always 1.

(i) To evaluate the impact of pattern graphs Q, we fixed $|V|$, *e.g.,* Amazon with 31245 nodes, YouTube with 9368 nodes, and synthetic data with 5×10^4 nodes, respectively, while varying $|V_q|$ from 2 to 20. Note that it took VF2 hours on all three datasets. Hence, we used randomly sampled smaller subgraphs of the original data graphs with hashing functions, commonly used in practical large-scale data process systems such as MapReduce [Dean and Ghemawat 2004] and Pregel [Malewicz et al. 2010].

(ii) To evaluate the impact of data graphs G, we fixed pattern graphs Q with $|V_q| = 10$ and varied the size of data graphs. We varied $|V|$ from 3×10^3 to 3×10^4 nodes for Amazon and from 10^3 to 10^4 for YouTube. For synthetic data, we varied $|V|$ from 10^4 to 10^5. We used relatively smaller data graphs since VF2 does not scale with large graphs.

The closeness results are reported in Figures 13(a), 13(b), 14(a), 14(b) 15(a), 15(b), 13(c), 13(d), 14(c), 14(d) 15(c), 15(d), 13(e), 13(f), 14(e), 14(f) 15(e), and 15(f). Note that



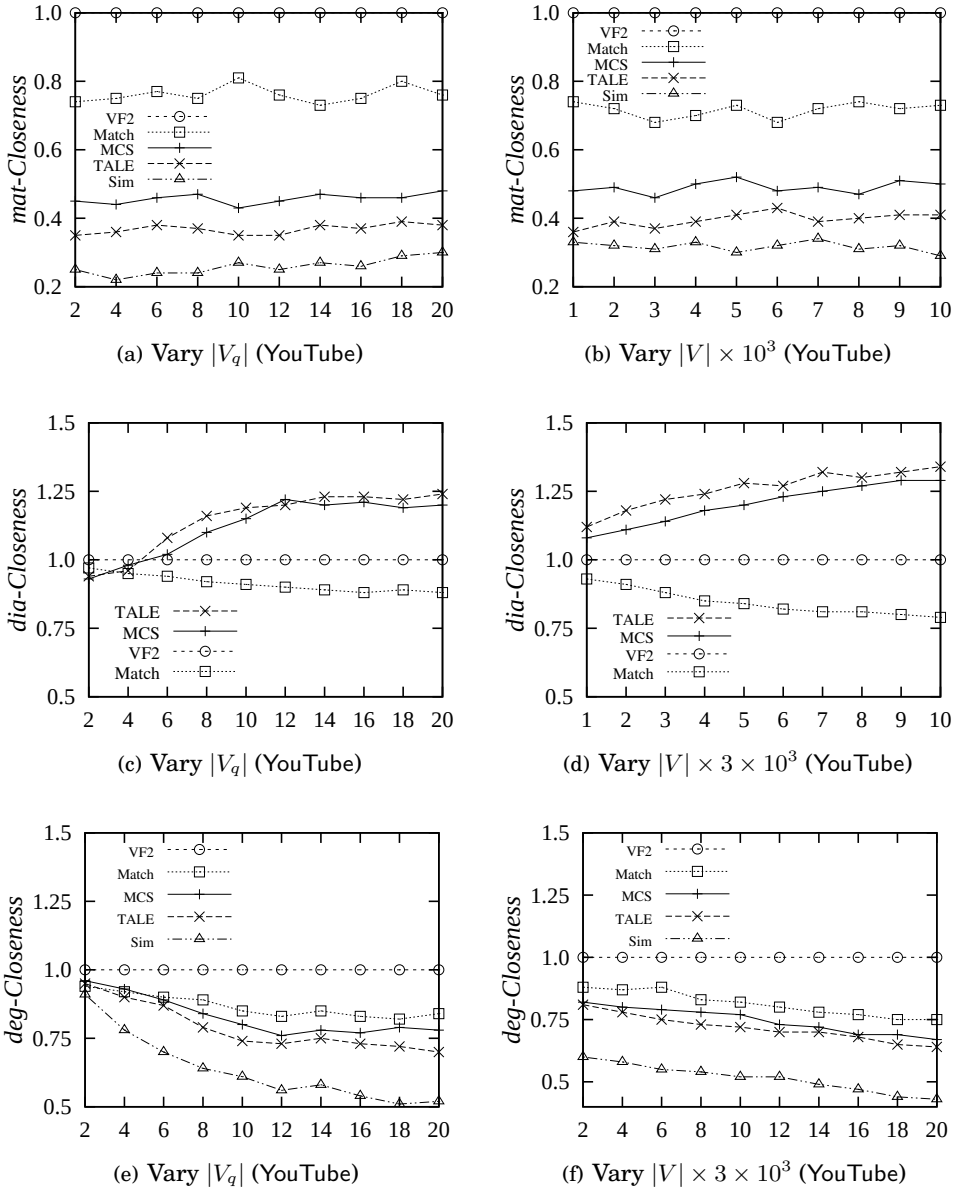
(a) Vary $|V_q|$ (Amazon) (b) Vary $|V| \times 3 \times 10^3$ (Amazon)

(c) Vary $|V_q|$ (Amazon) (d) Vary $|V| \times 3 \times 10^3$ (Amazon)

(e) Vary $|V_q|$ (Amazon) (f) Vary $|V| \times 3 \times 10^3$ (Amazon)

Fig. 13. Match quality evaluation on Amazon: closeness

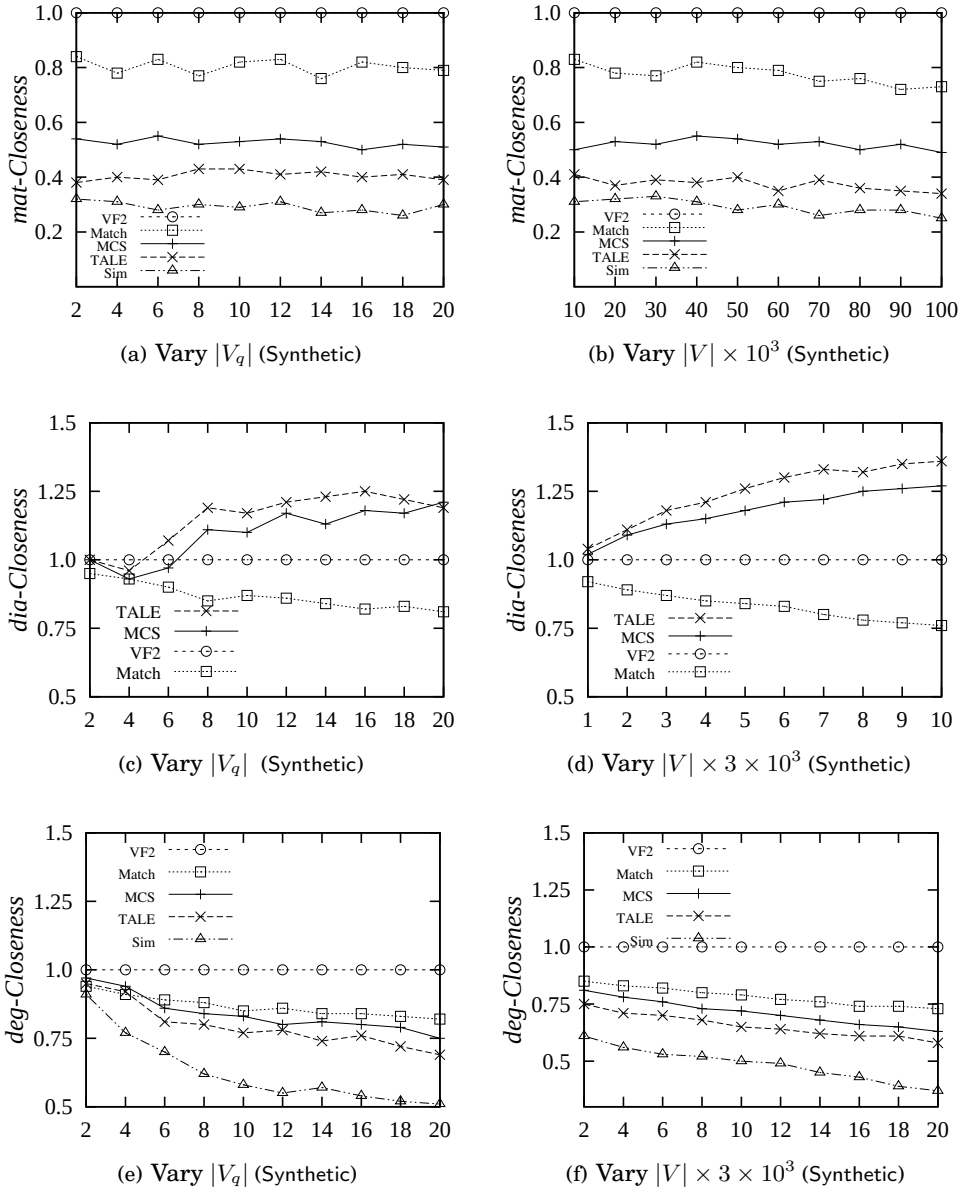
we did not report dia–closeness of Sim since in all cases the single matched subgraph returned is disconnected, *i.e.,* dia–closeness is treated as *zero* for all cases. Observe the following. (1) The mat–closeness of Match is consistently in the range of [70%, 80%] with various pattern and data graphs, while Sim is in [25%, 38%], TALE is in [35%, 42%], and MCS is in [46%, 57%], respectively. Hence, Match does much better than Sim (up to 50%), TALE (up to 36%) and MCS (up to 23%) for identifying matched nodes. Indeed, 70% to 80% of the matched nodes found by Match are exactly those found by VF2, which enforces strict topological matching. Recall that Match is able to find sensible matches



(a) Vary $|V_q|$ (YouTube)
(b) Vary $|V| \times 10^3$ (YouTube)
(c) Vary $|V_q|$ (YouTube)
(d) Vary $|V| \times 3 \times 10^3$ (YouTube)
(e) Vary $|V_q|$ (YouTube)
(f) Vary $|V| \times 3 \times 10^3$ (YouTube)

Fig. 14. Match quality evaluation on YouTube: closeness

missed by VF2 (Examples 1.1 and 2.1 and quality test (1) above). That is, the [20%, 30%] matches found by Match, but missed by VF2, further contain matched nodes that are sensible. (2) The dia–closeness of Match is consistently in the range of [75%, 96%] in all cases, while TALE is in [94%, 136%], and MCS is in [95%, 131%]. Thus Match is able to return matched subgraphs whose diameters are closer to the ones of the pattern graphs, compared to those matched subgraphs returned by TALE (up to 11%) and MCS (up to 6%). Here the dia–closeness of TALE and MCS is larger than 1 for the majority of cases since a large portion of matches found by them contain node mismatches and



(a) Vary $|V_q|$ (Synthetic) (b) Vary $|V| \times 10^3$ (Synthetic)

(c) Vary $|V_q|$ (Synthetic) (d) Vary $|V| \times 3 \times 10^3$ (Synthetic)

(e) Vary $|V_q|$ (Synthetic) (f) Vary $|V| \times 3 \times 10^3$ (Synthetic)

Fig. 15. Match quality evaluation on synthetic data: closeness

have much smaller diameters than the pattern graphs. (3) The deg–closeness of Match is consistently in the range of [77%, 95%] in all cases, while TALE is in [58%, 96%], MCS is in [61%, 94%], and Sim is in [38%, 89%]. This further shows that Match does better at preserving the degrees of pattern graphs than TALE (up to 19%), MCS (up to 16%) and Sim (up to 39%). These together tell us that Match is better at preserving structures of pattern graphs.

(3) In the same setting as (2) above for testing closeness, we tested the numbers of the

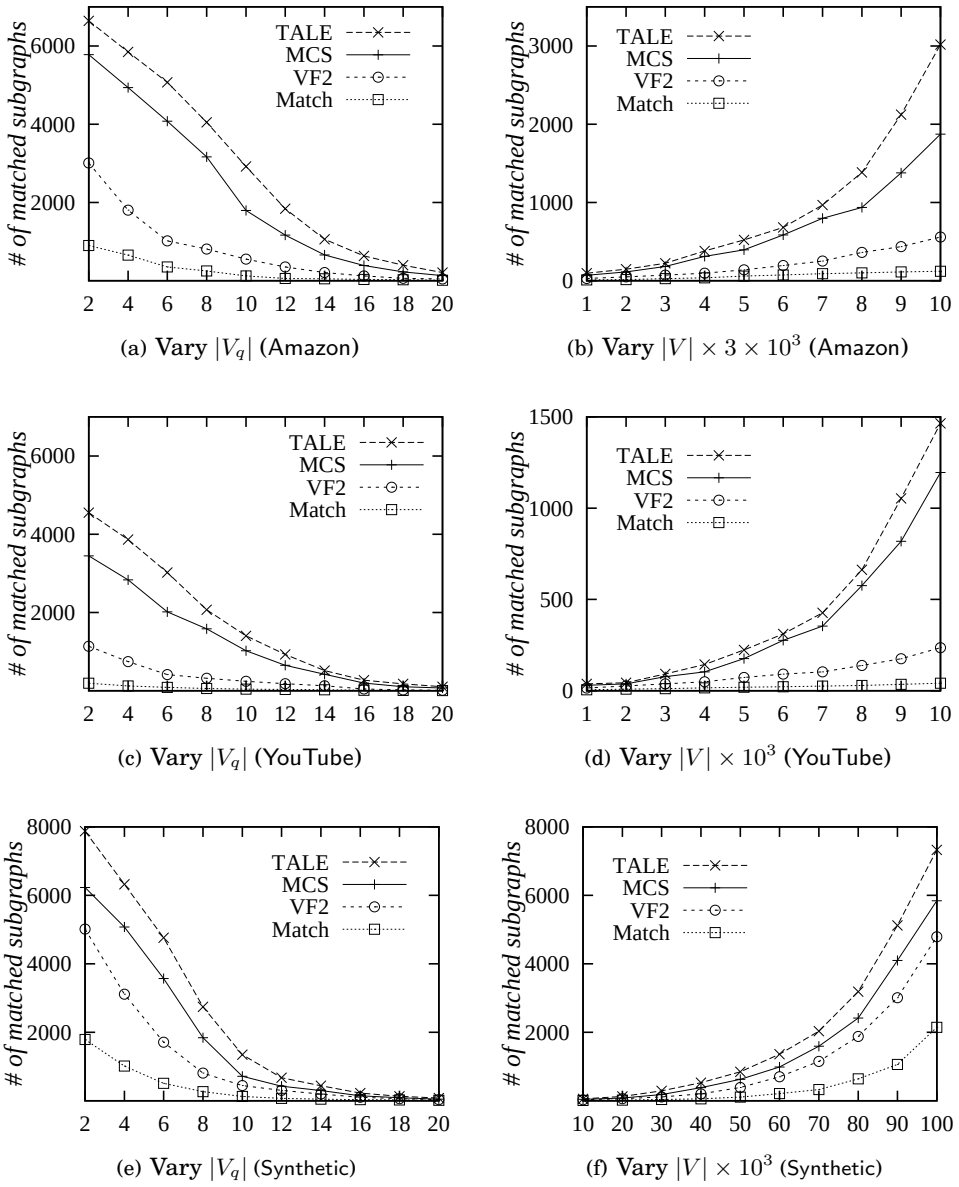


Fig. 16. Match quality evaluation: # *of matched subgraphs*

matched subgraphs in data graphs returned by Match, VF2, TALE and MCS. We did not report Sim since it always returns at most one matched subgraph.

The results are reported in Figures 16(a), 16(b), 16(c), 16(d), 16(e), and 16(f). They tell us that Match returns much less matched subgraphs than VF2: it returns consistently around 25% to 38% of matched subgraphs of VF2, for synthetic graph, Amazon and YouTube alike. For approximate matching algorithms TALE and MCS, it is obvious that they return much more subgraphs than VF2. Indeed, as shown in Fig. 16(f), for example, Match returns 2144 matched subgraphs compared to 4792 by VF2, 5843 by MCS

Table III. Match quality evaluation: sizes of matched subgraphs

#nodes	[0, 9]	[10, 19]	[20, 29]	[30, 39]	[40, 49]	≥ 50
Amazon	0	98	23	0	0	0
YouTube	0	21	18	1	1	0
Synthetic	0	187	113	65	6	0

and 7328 by TALE, on a synthetic data graph with 10^5 nodes. This confirms that Match effectively reduces the sizes of match results, and hence, allows users to effectively analyze the match results on large graphs in practice.

In addition, the number of matched subgraphs decreases when the size of pattern graphs increases, and it increases when the size of data graphs increases, as expected. We also find that although VF2 may find exponentially matches in theory, it does not happen very often in practice.

(4) In the same setting as (2) for testing closeness with smaller datasets, *e.g.,* Amazon with 31245 nodes, YouTube with 9368 nodes, and synthetic data with 100000 nodes, we tested the sizes of the matched subgraphs in data graphs returned by algorithms Match and Sim.

For Sim, it returns a single matched subgraph with 103, 177 and 311 nodes in Amazon, YouTube and synthetic data, respectively. For Match, the results are reported in Table III. Their matched subgraphs are typically small, where (a) all matched subgraphs have less than 50 nodes, and (b) over 80% of matches have less than 30 nodes, on real-life and synthetic data. This tells us that strong simulation indeed restricts the sizes of matches, due to the duality and locality.

Exp-2: Performance of centralized algorithms. In the second set of experiments, we evaluated the performance of our algorithms Match, Match$^+$ and algorithms Sim and VF2. Algorithm VF2 does not scale well with large data graphs, *e.g.,* it took VF2 more than *three* hours on data graphs with 5×10^6 nodes (when $\alpha = 1.2$). Hence, we only report the performance of VF2 on the small real-life datasets of Amazon and YouTube that were used to evaluate the quality of matches. For large synthetic data graphs, we only report the other three algorithms Match, Match$^+$ and Sim. In all of our experiments, we also found that TALE and MCS were even much slower than VF2, and hence we did not report the running time of TALE and MCS here.

(i) To evaluate the impact of pattern graphs Q, we used two small real-life datasets (Amazon and YouTube) and one large synthetic dataset. We fixed Amazon, YouTube and the synthetic data to have 3×10^4 nodes, 10^4 nodes and 5×10^6 nodes, respectively, while varying the number $|V_q|$ of pattern nodes from 2 to 20 or the density α_q of pattern graphs from 1.05 to 1.35 (*i.e.,* increasing the number of edges in pattern graphs).

(ii) To evaluate the impact of data graphs G, we used the same datasets as above. We fixed pattern graphs with $|V_q| = 10$, while varying the number $|V|$ of nodes of Amazon, YouTube and the synthetic data from 6×10^3 to 3×10^4, 2×10^3 to 10^4 and 10^6 to 10^7, respectively, or varying the density α of data graphs from 1.05 to 1.35.

In the settings of (i) and (ii), we evaluated the running time of the algorithms concerned. We report our findings below.

(1) The impacts of pattern graphs on the elapsed time of algorithms VF2, Match, Match$^+$ and Sim are reported in Figures 17(a), 17(b), 17(c) and 17(d) for real-life datasets and in Figures 17(e) and 17(f) for synthetic datasets, respectively. Observe the following.

(a) When varying $|V_q|$. (i) As shown in Figures 17(a) and 17(c),VF2 is consistently much slower than the other three algorithms on both Amazon and YouTube, It is about 100

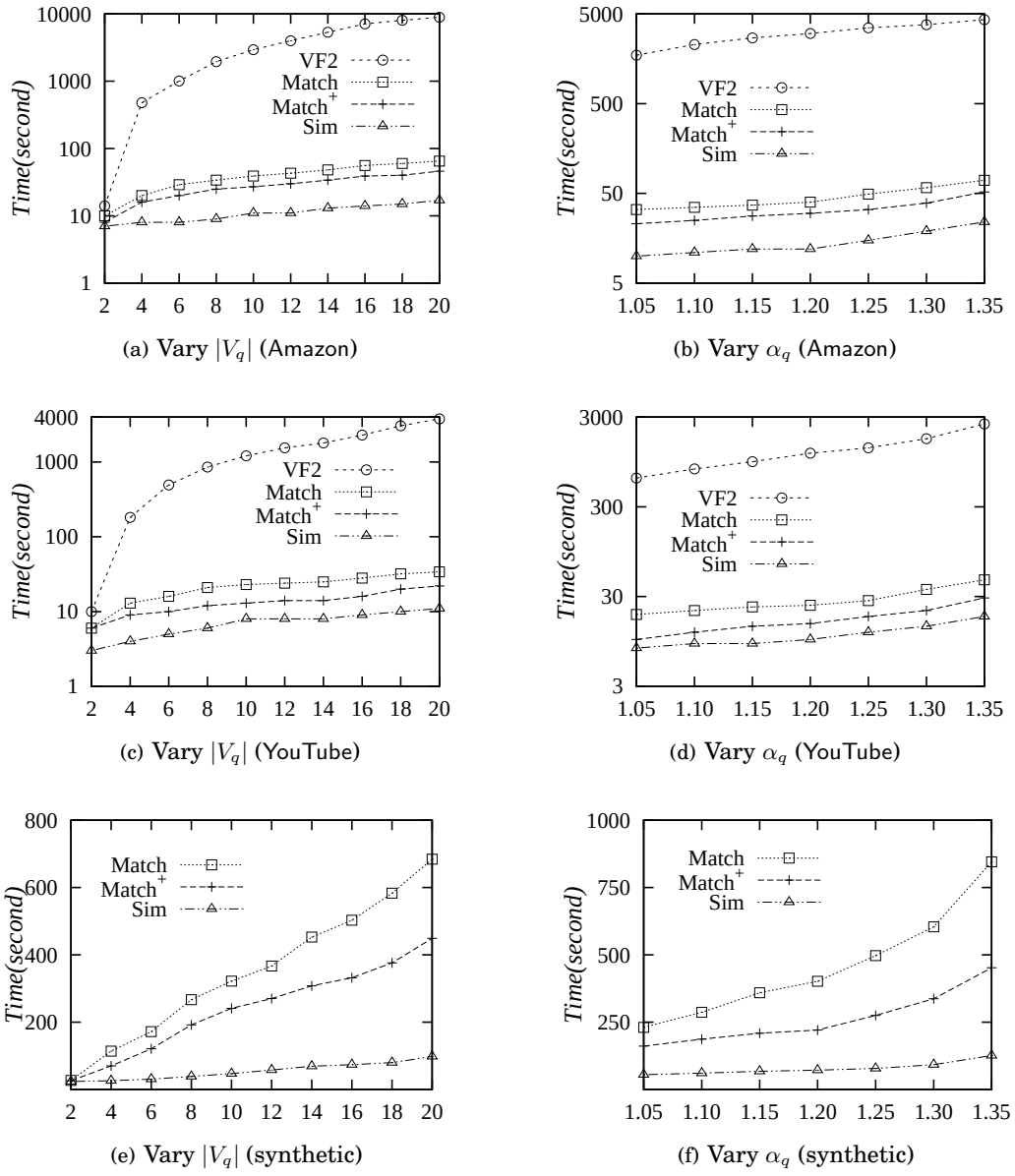
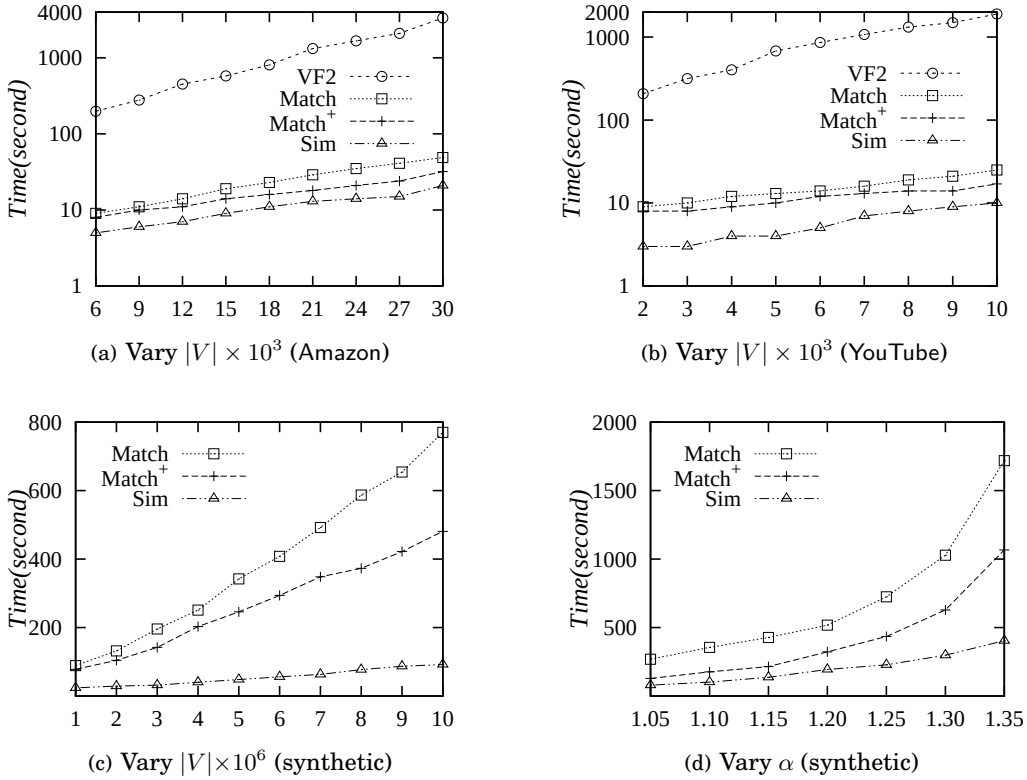


Fig. 17. Performance evaluation of centralized algorithms: vary pattern graphs

times slower than Match$^+$ when $V_q \geq 4$ on the two real-life datasets. For instance, it took VF2 hours on Amazon and YouTube. Note that, however, when $|V_q| = 2$, VF2 is almost as efficient as the other algorithms. This is consistent with the complexity analysis of VF2: VF2 is in low PTIME when $|V_q| = 2$. (ii) As shown in Fig. 17(e), all these algorithms scale well with $|V_q|$ on large data graphs, except VF2.

(b) When varying the density α_q of pattern graphs. (i) Figures 17(b) and 17(d) show that on real-life datasets (Amazon and YouTube), VF2 is consistently much slower than



(a) Vary $|V| \times 10^3$ (Amazon)

(b) Vary $|V| \times 10^3$ (YouTube)

(c) Vary $|V| \times 10^6$ (synthetic)

(d) Vary α (synthetic)

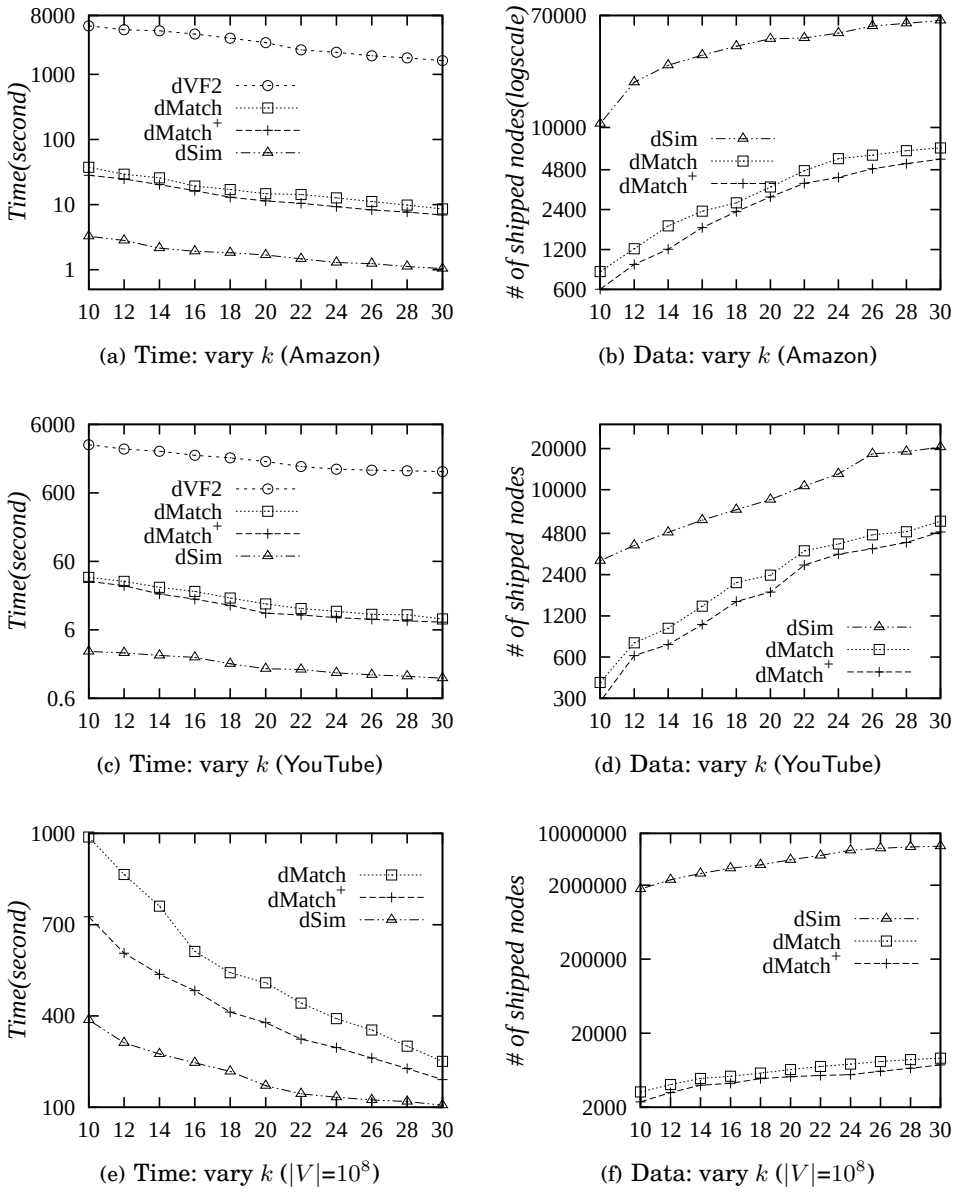
Fig. 18. Performance evaluation of centralized algorithms: vary data graphs

the other three algorithms on both Amazon and YouTube, which is similar to the case when varying $|V_q|$ on real-life datasets. Indeed, the running time of VF2 is consistently over 1700s for Amazon while it is always under 80s for the other three algorithm. (ii) Figure 17(f) shows that that these algorithms scale well with the density α_q on large data graphs, except VF2. Algorithms Match and Match$^+$ are slower than Sim, as expected. Indeed, this is a price that has to be paid in exchange for better match quality. We did not report the performance of VF2 in Figures 17(e) and 17(f) since it could not run to completion when $|V_q| \geq 4$.

Finally, observe that the running time of all algorithms increases when $|V_q|$ or α_q increases. This is consistent with the complexity analyses of these algorithms.

(2) The impacts of data graphs on the running time of algorithms VF2, Match, Match$^+$ and Sim are shown in Figures 18(a) and 18(b) for real-life datasets and Figures 18(c) and 18(d) for synthetic datasets.

These results are consistent with the results of varying pattern graphs. (a) As shown in these figures, all these algorithms except VF2 scale well with the size of data graphs and with the density α of data graphs; (b) algorithms Match and Match$^+$ are slower than Sim; and (c) the running time of VF2 increases far more substantially with the size and density of data graphs than the others. For example, the running time of Match$^+$ increased from about 100s to 600s when the number of nodes of the synthetic data varied from 10^6 to 10^7; in contrast, VF2 spent nearly 4000s on Amazon data with 3×10^4 nodes, but only around 30s on Amazon graphs with 3×10^3 nodes.



(a) Time: vary k (Amazon)

(b) Data: vary k (Amazon)

(c) Time: vary k (YouTube)

(d) Data: vary k (YouTube)

(e) Time: vary k ($|V|$=10^8)

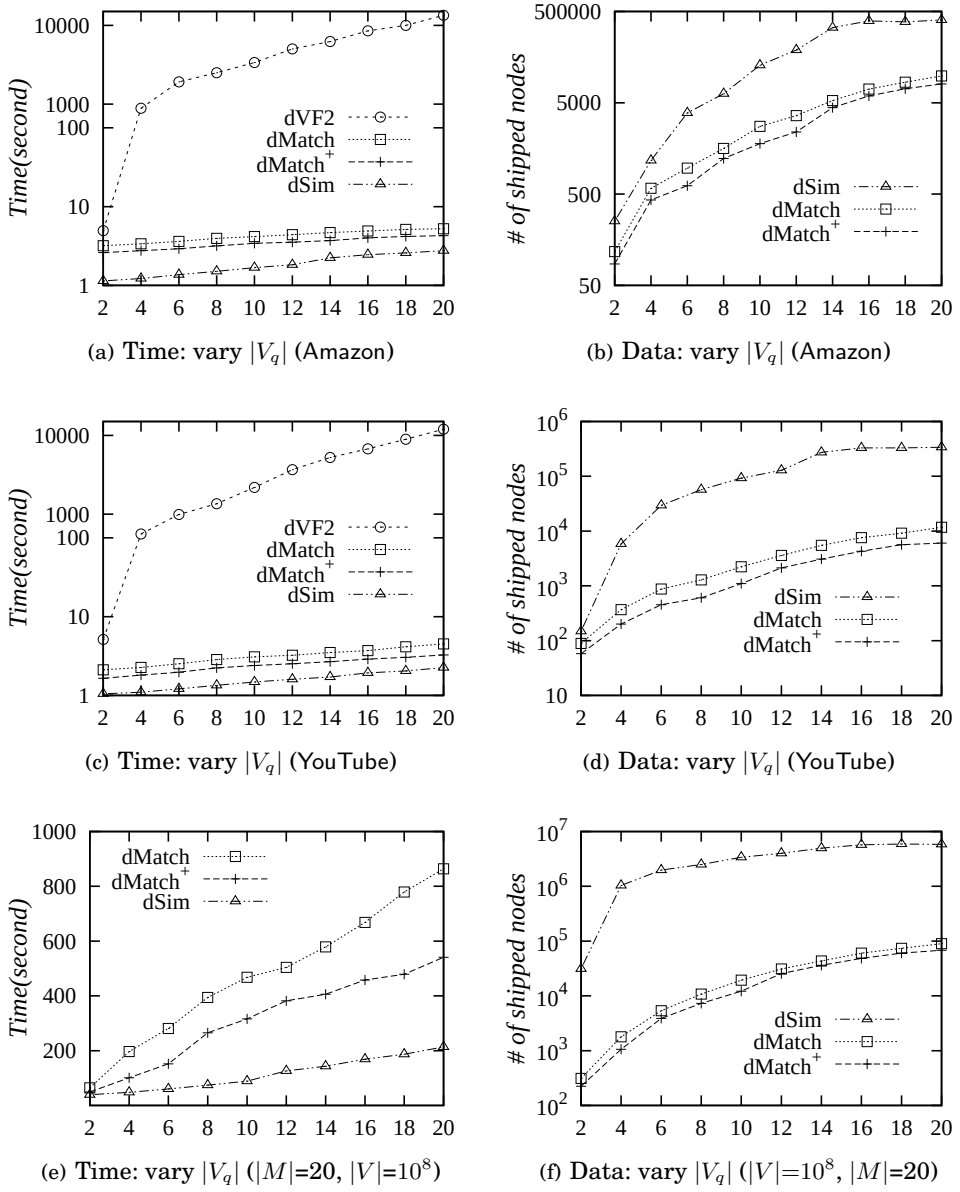
(f) Data: vary k ($|V|$=10^8)

Fig. 19. Performance evaluation of distribute algorithms: vary k

(3) The experimental results in (1) and (2) above also verify that our optimization techniques are effective. Indeed, the running time of Match$^+$ is consistently about $2/3$ of the time taken by Match, a significant improvement.

These results tell us that all algorithms except VF2 scale well *w.r.t.* large data graphs on single machines, and the optimization techniques are effective.

Exp-3: Performance of distributed algorithms. In the last set of experiments, we evaluated the efficiency and data shipment of our distributed algorithms dMatch,



(a) Time: vary $|V_q|$ (Amazon)

(b) Data: vary $|V_q|$ (Amazon)

(c) Time: vary $|V_q|$ (YouTube)

(d) Data: vary $|V_q|$ (YouTube)

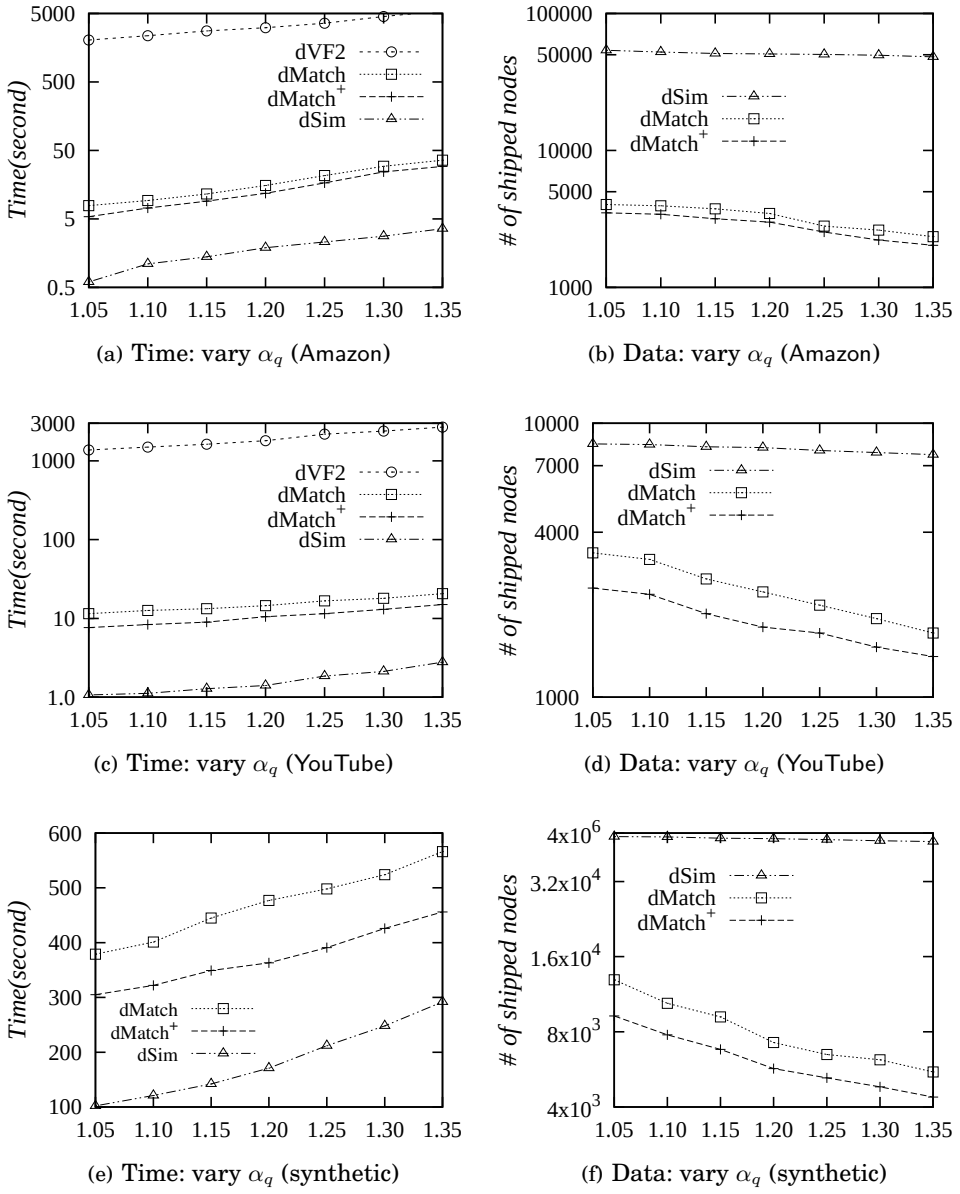
(e) Time: vary $|V_q|$ ($|M|$=20, $|V|$=10^8)

(f) Data: vary $|V_q|$ ($|V|$=10^8, $|M|$=20)

Fig. 20. Performance evaluation of distributed algorithms: vary $|V_q|$

dMatch$^+$ and algorithms dSim and dVF2. All the datasets are partitioned with a modulo hashing function: hash(id) mod k, where id is the identifier of a node and k is the number of participating machines. This partitioning approach has been commonly adopted in large-scale data processing systems, such as MapReduce [Dean and Ghemawat 2004] and Pregel [Malewicz et al. 2010]. Here we did not report the performance of algorithm dVF2 on large synthetic graphs as dVF2 did not run to completion in this case.

(i) To evaluate the impacts of the number k of machines, we fixed pattern graphs with



(a) Time: vary α_q (Amazon)

(b) Data: vary α_q (Amazon)

(c) Time: vary α_q (YouTube)

(d) Data: vary α_q (YouTube)

(e) Time: vary α_q (synthetic)

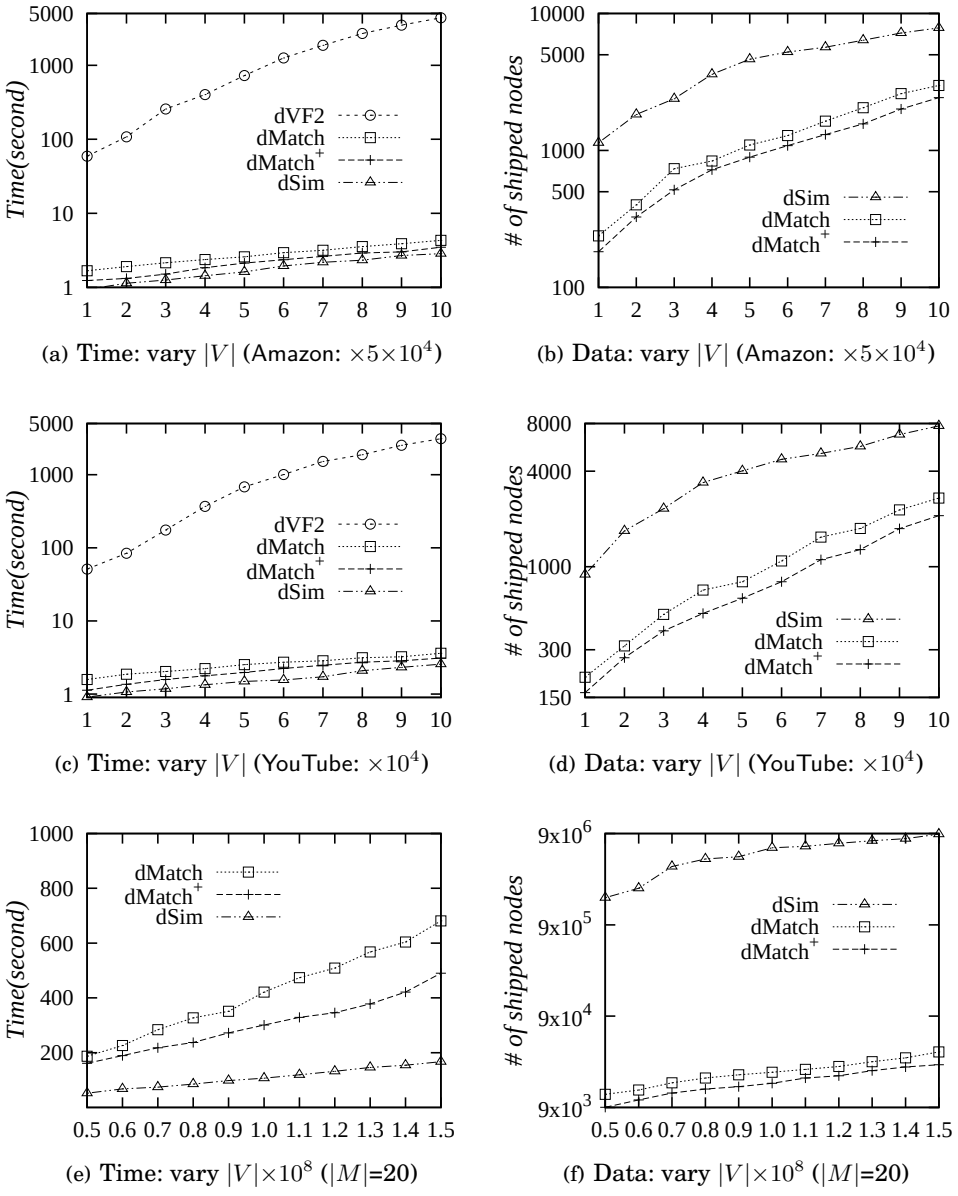
(f) Data: vary α_q (synthetic)

Fig. 21. Performance evaluation of distributed algorithms: vary α_q

$|V_q| = 10$ and $\alpha_q = 1.2$, and data graphs with $|V| = 5 \times 10^5$, 10^5 and 10^8 for Amazon, YouTube and synthetic data, respectively, and $\alpha = 1.2$, while varying k from 10 to 30.

(ii) To evaluate the impacts of pattern graphs Q, we fixed k = 20 and data graphs using the same setting as (i), while varying $|V_q|$ from 2 to 20 or α_q from 1.05 to 1.35.

(iii) To evaluate the impacts of data graphs G, we fixed $|V_q| = 10$ and $k = 20$, while varying $|V|$ from 0.5×10^5 to 5.0×10^5, 0.1×10^5 to 1.0×10^5 and 0.5×10^8 to 1.5×10^8 for Amazon, YouTube and synthetic data, respectively, or α from 1.05 to 1.35.



(a) Time: vary $|V|$ (Amazon: $\times 5 \times 10^4$)

(b) Data: vary $|V|$ (Amazon: $\times 5 \times 10^4$)

(c) Time: vary $|V|$ (YouTube: $\times 10^4$)

(d) Data: vary $|V|$ (YouTube: $\times 10^4$)

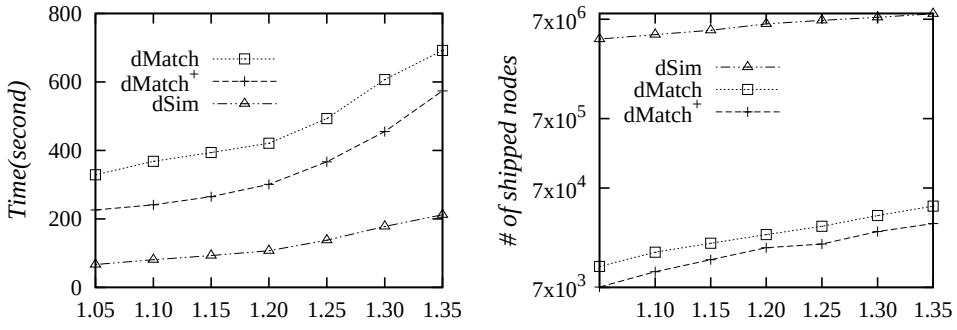
(e) Time: vary $|V| \times 10^8$ ($|M|$=20)

(f) Data: vary $|V| \times 10^8$ ($|M|$=20)

Fig. 22. Performance evaluation of distributed algorithms: vary $|V|$

In the settings of (i), (ii) and (iii), we evaluated the elapsed time and the data shipment (*i.e.,* the number of shipped nodes) for those distributed algorithms concerned. We report our findings as follows.

(1) The results on the running time of algorithms dVF2, dMatch, dMatch$^+$ and dSim are reported in Figures 19(a), 19(c), 20(a), 20(c), 21(a), 21(c), 22(a) and 22(c) for real-life datasets (Amazon and YouTube) and Figures 19(e), 20(e), 21(e), 22(e) and 23(a) for synthetic datasets, respectively. One can observe the following in these figures.



(a) Time: vary α (b) Data: vary α

Fig. 23. Performance evaluation of distributed algorithms: vary α

(a) On all the datasets, the running time of all algorithms decreases *w.r.t.* the number k of participating machines, as expected.

(b) Our algorithms dMatch and dMatch$^+$ are slower than dSim. This is easy to understand since graph simulation is solvable in quadratic time, while strong simulation is solvable in cubic time. Moreover, the data shipment is fast and takes a small portion of time in a cluster of machines. However, as shown in Figures 19(a), 19(c) and 19(e), the more machines are used, the more benefits that dMatch and dMatch$^+$ obtain. Indeed, the running time of dMatch increased over 4 times when the number k of machines used decreased from 30 to 10, while dSim increased only 2.4 times when k varied in the same way. This means our distributed algorithms are more capable of exploring the advantages of distributed computing paradigm, which is consistent with the locality analysis of strong simulation.

(c) The running time of all algorithms increases *w.r.t.* the pattern graph sizes (*i.e.,* $|V_q|$ and α_q) and data graph sizes (*i.e.,* $|V|$ and α), as shown in Figures 20(a), 20(c), 20(e) and 21(a), 21(c), 21(e).

(d) Figures 19(e), 20(e), 21(e), 22(e) and 23(a) tell us that all the algorithms except dVF2 scale well in all cases. When data graphs are large, dVF2 is about 1000 times slower than the other algorithms.

(e) Our optimization techniques are effective: dMatch$^+$ takes about $2/3$ to $3/4$ of the running time of dMatch on large synthetic data, which is consistent with the results of the centralized setting. For real-life data, dMatch$^+$ takes about [78%, 85%] and [70%, 77%] of the running time of dMatch on Amazon and YouTube, respectively. Indeed, dMatch$^+$ is efficient, *e.g.,* it only took 270s for data graphs with $|V| = 10^8$, pattern graphs with $|V_q| = 10$ and $k = 30$. Note that dMatch$^+$ has less advantage than dMatch on Amazon since the number of boundary nodes of a fragmented Amazon is relatively small. Indeed, Amazon is a relatively sparse graph with its α = 1.08 (recall that $m = n^\alpha$ for a graph with n nodes and m edges). For YouTube, it is much better as its α is around 1.2. In addition, the edges of both Amazon and YouTube form small dense clusters that are connected sparsely, which further reduces the number of boundary nodes.

(2) The results on the the number of shipped nodes of algorithms dMatch, dMatch$^+$ and dSim are reported in Figures 19(b), 19(d), 20(b), 20(d), 21(b), 21(d), 22(b), and 22(d) for real-life datasets (Amazon and YouTube) and Figures 19(f), 20(f), 21(f), 22(f), and 23(b) for synthetic datasets, respectively. Note that dVF2 shipped the same amount of data

as dMatch did in all datasets. Hence we do not report the number of shipped nodes of dVF2 here. One can observe the following from these figures.

(a) The shipped data increases when k increases for both real-life and synthetic data, as shown in Figures 19(b), 19(d) and 19(f). Indeed, the data graph is partitioned and distributed across k machines. Since the number of boundary nodes increases with the increase of the number of partitions, the shipped data increases as well. Nevertheless, the amount of data shipped is low: for example, it accounts for around 10^{-2} and 10^{-3} to 10^{-4} of the entire data graph when k = 30 for real-life and synthetic data sets, respectively, and it is even 4×10^{-4} for synthetic data with 10^8 nodes.

(b) The shipped data increases when the number $|V_q|$ of pattern graphs increases, as shown in Figures 20(b), 20(d) and 20(f). When pattern graphs are larger, so are their diameters; hence the increase in the amount of data shipped. Nonetheless, the shipped data accounts for only 10^{-3} of the entire synthetic data even when $|V_q| = 20$.

(c) The shipped data of our algorithms dMatch and dMatch$^+$ decreases when the density α_q of pattern graphs increases, as shown in Figures 21(b), 21(d) and 21(f). Indeed, when the number $|V_q|$ of nodes in the pattern graphs is fixed, the larger α_q is, the smaller their diameters are. Hence the amount of data shipped decreases. Different from dMatch and dMatch$^+$, dSim is not very sensitive to the density of pattern graphs.

(d) The data shipped increases when data graphs get larger and denser, as shown in Figures 22(b), 22(d), 22(f) and 23(b). Indeed, the larger or denser the data graphs are, the more boundary nodes there are, and hence, the more data shipped. Again, the amount of data shipment is indeed rather small, compared to the entire datasets, *e.g.,* data shipped only accounts for 10^{-3} of the two real-life datasets, and is only 3.5×10^{-4} for synthetic graphs with $|V| = 1.5 \times 10^8$.

(e) Our algorithms dMatch and dMatch$^+$ shipped much less nodes than dSim on all datasets, *e.g.,* the number of nodes shipped by dSim was about [15, 20] times larger than those by dMatch$^+$ on YouTube when $|V_q| = 10$.

(f) Our optimization techniques are effective, which reduce the amount of data shipment, since dMatch$^+$ consistently shipped less nodes than dMatch on all datasets, especially on those large and dense synthetic data graphs. For example, dMatch$^+$ shipped only 70% of those nodes shipped by dMatch on synthetic data when $|V| = 1.5 \times 10^8$ and $\alpha_q = 1.05$. Similar to the case for testing the running time, dMatch$^+$ has much more significant advantage over dMatch on large and dense synthetic data graphs than spatial real-life data graph Amazon, for the same reason.

Summary. From these experimental results we find the following. (1) Strong simulation is able to identify sensible matches that are not found by subgraph isomorphism, and eliminate insensible matches found by graph simulation. In addition, it finds high quality matches that retain graph topology. Indeed, 70%-80% of matches found by subgraph isomorphism are retrieved by strong simulation, (up to 50%) better than graph simulation, without paying the price of intractable complexity and large number (or size) of matches. (2) Our algorithms for strong simulation, centralized or distributed, are efficient and scale well with the size and density large-scale data graphs, *e.g.,* it took 270 seconds when $|V| = 10^8$, $|V_q| = 10$ and $|M| = 30$. (3) Our optimization techniques are effective, reducing the running time by at least 25%, 23% and 15% on synthetic data, YouTube and Amazon, respectively. (4) The locality of strong simulation allows efficient distributed evaluation algorithms, which incur network overhead of only 10^{-2} to 10^{-4} of the entire data graphs.

7. RELATED WORK

Strong simulation was introduced in [Ma et al. 2011]. This article extends [Ma et al. 2011] by including (a) proofs for all the results; (b) optimization techniques for the distributed algorithm of strong simulation (Section 5.2); and (c) an extensive experimental study compared to the preliminary study of [Ma et al. 2011] (Section 6). We remark that strong simulation preserves the topology of graphs and has the same complexity as earlier extensions [Fan et al. 2010a; Fan et al. 2011] of graph simulation [Milner 1989].

There has been a host of work on graph pattern matching via subgraph isomorphism (*e.g.,* [Tian and Patel 2008; Tong et al. 2007; Zou et al. 2009]; see [Aggarwal and Wang 2010; Gallagher 2006] for surveys). In light of its intractability, approximate matching has been studied to find inexact solutions, which allows node/edge mismatches [Aggarwal and Wang 2010; Tian and Patel 2008]. This work differs from approximate matching in that no node/edge mismatches are allowed, and that the number of matches via strong simulation is linear in the size of the data graph rather than exponential for (approximate) subgraph isomorphism. Extensions of subgraph isomorphism are studied in [Fan and Bohannon 2008; Fan et al. 2010b; Zou et al. 2009], which extend mappings from edge-to-edge to edge-to-path. Nevertheless, these problems remain NP-complete.

Closer to this work are bounded simulation [Fan et al. 2010a] and graph pattern queries of [Fan et al. 2011]. The former extends graph simulation [Milner 1989] by allowing bounds on the number of hops in pattern graphs, and the latter further extends [Fan et al. 2010a] by incorporating regular expressions as edge constraints on pattern graphs. Graph pattern matching via these extensions are in cubic-time [Fan et al. 2010a; Fan et al. 2011]. As remarked earlier, these notions of graph simulation may fail to capture the topology of graphs, and yield false matches or too large a match relation. These are precisely the problems that strong simulation aims to rectify, by imposing additional constraints (duality and locality) on graph simulation.

Restricting search in a confined space has been adopted by keyword search on graphs [Li et al. 2008; Qin et al. 2009; Kargar and An 2011], in which the diameter of the identified subgraphs is bounded by a parameter r, determined by experts. Similarly, the data locality of strong simulation restricts the search space of a match graph in a ball. However, the radius of balls is determined by pattern graphs only, and, hence there is no need for any prior knowledge to explicitly set up such a parameter.

Schema extraction is to discover the implicit structure of semi-structured data, which has no schema predefined. It has proved effective in query formulation and optimization [Abiteboul et al. 1999; Goldman and Widom 1997]. Schema of semi-structured data is often extracted via a mild generalization of graph simulation that deals with labeled edges [Abiteboul et al. 1999]. Nevertheless, topology preservation is not an issue in schema extraction, and no previous work there has studied how graph simulation should be refined to capture topology.

Query minimization, as a classical optimization technique, has been well studied for SQL queries [Abiteboul et al. 1995], XPath (*e.g.,* [Chen and Chan 2008]), graph simulation [Bustan and Grumberg 2003] and graph pattern queries [Fan et al. 2011]. This work explores it for graph pattern matching via strong simulation.

Distributed query processing has been studied for relational data [Kossmann 2000] and XML [Cavendish and Candan 2008; Cong et al. 2007]. There has also been recent work on distributed graph processing to manage large-scale graphs [Dean and Ghemawat 2004; Giatsoglou et al. 2011; Malewicz et al. 2010]. However, to the best of our knowledge, (a) the only prior methods for distributed computation of graph simulation [Milner 1989] are [Ma et al. 2012] and [Fan et al. 2012b] for restricted pattern

queries, and (b) no previous work has studied distributed computation of its extensions [Fan et al. 2010a; Fan et al. 2011], not to mention strong simulation proposed in this work.

In this work, we assume that data graphs are already partitioned. Indeed, how data graphs are partitioned may have a significant impact on the evaluation of strong simulation. Graph partitioning is a traditional problem that has been extensively studied since 1970's [Kernighan and Lin 1970; Karypis and Kumar 1998; Yang et al. 2012]. It is to find a set of non-overlapping fragments for a given graph such that (a) all fragments have a roughly equal number of nodes, and (b) the number of edges connecting nodes in different fragments is minimized. Although graph partitioning is an NP-hard problem [Garey and Johnson 1979], large-scale graph partitioning tools are available such as the well-known METIS [Karypis and Kumar 1998]. A refined partition of data graphs could certainly benefit the computation of strong simulation. Hence, the prior work is essentially orthogonal, but complementary, to this work.

8. CONCLUSION

We have proposed strong simulation to rectify problems of graph pattern matching based on subgraph isomorphism and graph simulation. We have verified, both analytically and experimentally, that strong simulation has several salient features, notably (1) it is capable of capturing the topological structures of pattern and data graphs; (2) it retains the same cubic-time complexity as previous extensions of graph simulation, (3) it demonstrates data locality and allows efficient distributed evaluation algorithms, and (4) it finds bounded matches. Our experimental results have also verified the effectiveness of our optimization techniques.

Several topics are targeted for future work. First, we are to extend strong simulation by incorporating regular expressions on edge types, along the same lines as [Fan et al. 2011]. Second, our distributed algorithms just aim to demonstrate the data locality of strong simulation. More sophisticated algorithms can be developed in the distributed setting, with better performance guarantees. Finally, for large graphs, cubic time is still too expensive. We are to explore new techniques to speed up the computation. In particular, we are investigating the following strategies: (lossy) graph compression schemes that preserves strong simulation [Fan et al. 2012a], view-based graph pattern matching [Halevy 2001], metrics to rank match graphs and to return top-ranked match graphs without computing the entire set of all matches [Fan et al. 2013a], incremental methods for strong simulation to minimize unnecessary recomputation in response to (typically frequent) changes to real-life graphs [Fan et al. 2013b], and distributed algorithms in the GraphLab model [Low et al. 2010]. We expect that combinations of these strategies will yield effective and efficient methods for computing strong simulation in large real-life graphs. When necessary, inexact algorithms should be developed to compute matches in big graphs, ideally with certain performance guarantees.

ACKNOWLEDGMENTS

Ma is supported in part by NSFC grant 61322207, NGFR 973 grant 2014CB340304 and 863 grant 2013AA01A213. Fan is supported in part by the 973 Programs 2012CB316200 and 2014CB340302, Guangdong Innovative Research Team Program 2011D005 and the Shenzhen Peacock Program 1105100030834361 of China, as well as EPSRC EP/J015377/1, UK.

References

Serge Abiteboul. 1997. Querying Semi-Structured Data. In *ICDT*.

Serge Abiteboul, Peter Buneman, and Dan Suciu. 1999. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann.

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

Charu C. Aggarwal and Haixun Wang. 2010. *Managing and Mining Graph Data*. Springer.

Sihem Amer-Yahia, Michael Benedikt, and Philip Bohannon. 2007. Challenges in Searching Online Communities. *IEEE Data Eng. Bull.* 30, 2 (2007), 23–31.

Joel Brynielsson, Johanna Hogberg, Lisa Kaati, Christian Mårtenson, and Pontus Svenson. 2010. Detecting Social Positions Using Simulation. In *ASONAM*.

Nancy Buchan and Rachel Croson. 2004. The boundaries of trust: own and others' actions in the US and China. *Journal of Economic Behavior & Organization* 55, 4 (2004), 485–504.

Doron Bustan and Orna Grumberg. 2003. Simulation-based minimazation. *ACM Trans. Comput. Log.* 4, 2 (2003), 181–206.

Dirceu Cavendish and K. Selçuk Candan. 2008. Distributed XML processing: Theory and applications. *J. Parallel Distrib. Comput.* 68, 8 (2008), 1054–1069.

Alan Chia-Lung Chen, Shang Gao, Panagiotis Karampelas, Reda Alhajj, and Jon G. Rokne. 2011. Finding Hidden Links in Terrorist Networks by Checking Indirect Links of Different Sub-Networks. In *Counterterrorism and Open Source Intelligence*. 143–158.

Ding Chen and Chee Yong Chan. 2008. Minimization of tree pattern queries with constraints. In *SIGMOD*.

Gao Cong, Wenfei Fan, and Anastasios Kementsietsidis. 2007. Distributed query evaluation with performance guarantees. In *SIGMOD*.

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 10 (2004), 1367–1372.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. The MIT Press.

G. Csardi and T. Nepusz. 2006. The igraph software package for complex network research. *InterJournal Complex Systems* 1695 (2006).

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*.

R. Diestel. 2005. *Graph Theory*. Springer.

Agostino Dovier and Carla Piazza. 2003. The Subgraph Bisimulation Problem. *IEEE Trans. Knowl. Data Eng.* 15, 4 (2003), 1055–1056.

Wenfei Fan and Philip Bohannon. 2008. Information Preserving XML Schema Embedding. *TODS* 33, 1 (2008).

Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. 2011. Adding Regular Expressions to Graph Reachability and Pattern Queries. In *ICDE*.

Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. 2010a. Graph Pattern Matching: From Intractable to Polynomial Time. *PVLDB* 3, 1 (2010), 264–275.

Wenfei Fan, Jianzhong Li, Shuai Ma, Hongzhi Wang, and Yinghui Wu. 2010b. Graph Homomorphism Revisited for Graph Matching. *PVLDB* 3, 1 (2010), 1161–1172.

Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. 2012a. Query Preserving Graph Compression. In *SIGMOD*.

Wenfei Fan, Xin Wang, and Yinghui Wu. 2012b. Performance Guarantees for Distributed Reachability Queries. *PVLDB* 5, 11 (2012), 1304–1315.

Wenfei Fan, Xin Wang, and Yinghui Wu. 2013a. Diversified Top-k Graph Pattern Matching. *PVLDB* (2013).

Wenfei Fan, Xin Wang, and Yinghui Wu. 2013b. Incremental Graph Pattern Matching. *TODS* (2013). to appear.

Arash Fard, Amir Abdolrashidi, Lakshmish Ramaswamy, and John A. Miller. 2012. Towards efficient query processing on massive time-evolving graphs. In *CollaborateCom*.

Brian Gallagher. 2006. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS.* (2006).

Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Maria Giatsoglou, Symeon Papadopoulos, and Athena Vakali. 2011. Massive Graph Management for the Web and Web 2.0. In *New Directions in Web Data Management 1*. Springer.

Roy Goldman and Jennifer Widom. 1997. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB*.

Martin Grohe. 2010. From polynomial time queries to graph structure theory. In *ICDT*.

Alon Y. Halevy. 2001. Answering queries using views: A survey. *VLDB J.* 10, 4 (2001), 270–294.

M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. 1995. Computing simulations on finite and infinite graphs. In *FOCS*.

Viggo Kann. 1992. On the Approximability of the Maximum Common Subgraph Problem. In *STACS*.

M. Kargar and A. An. 2011. Keyword search in graphs: finding r-cliques. *PVLDB* 4, 10 (2011), 681–692.

George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SISC* 20, 1 (1998), 359–392.

Brian W Kernighan and S. Lin. 1970. An efficientheuristic procedure for partitioning graphs. *Bell System Technical Journal* 49, 1 (1970), 13–21.

Arijit Khan, Yinghui Wu, Charu C Aggarwal, and Xifeng Yan. 2013. NeMa: fast graph search with label similarity. *PVLDB* 6, 3 (2013), 181–192.

Donald Kossmann. 2000. The State of the art in distributed query processing. *ACM Comput. Surv.* 32, 4 (2000), 422–469.

Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. 2008. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*.

David Liben-Nowell and Jon M. Kleinberg. 2003. The link prediction problem for social networks. In *CIKM*.

Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. 2006. GPLAG: detection of software plagiarism by program dependence graph analysis. In *KDD*.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. GraphLab: A New Parallel Framework for Machine Learning. In *UAI*.

Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. 2011. Capturing Topology in Graph Pattern Matching. *PVLDB* 5, 4 (2011), 310–321.

Shuai Ma, Yang Cao, Jinpeng Huai, and Tianyu Wo. 2012. Distributed graph pattern matching. In *WWW*.

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *SIGMOD*.

Robin Milner. 1989. *Communication and Concurrency*. Prentice Hall.

Christos H Papadimitriou. 1994. *Computational Complexity*. Addison-Wesley.

L. Qin, J.X. Yu, L. Chang, and Y. Tao. 2009. Querying communities in relational databases. In *ICDE*.

Einat Sprinzak, Shmuel Sattath, and Hanah Margalit. 2003. How reliable are experimental protein–protein interaction data? *Journal of molecular biology* 327, 5 (2003), 919C923.

Loren G. Terveen and David W. McDonald. 2005. Social matching: A framework and research agenda. In *ACM Trans. Comput.-Hum. Interact.* 401–434.

Yuanyuan Tian and Jignesh M. Patel. 2008. TALE: A Tool for Approximate Large Graph Matching. In *ICDE*.

Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. 2007. Fast best-effort pattern matching in large attributed graphs. In *KDD*.

Julian R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *J. ACM* 23, 1 (1976), 31–42.

Shengqi Yang, Xifeng Yan, Bo Zong, and Arijit Khan. 2012. Towards effective partition management for large graphs. In *SIGMOD*.

Lei Zou, Lei Chen, and M. Tamer Özsu. 2009. DistanceJoin: Pattern Match Query In a Large Graph Database. *PVLDB* 2, 1 (2009), 886–897.

Received January 2013; revised May 2013; accepted September 2013